TULIPP

H2020-ICT-O4-2015

Grant Agreement n° 688403

D1.2: Reference Platform v2

Authors

Organization	Participant
THALES	Philippe Millet
SUNDANCE	Flemming Christensen
IOSB	Igor Tchouchenkov
HIPPEROS	Antonio Paolillo
NTNU	Magnus Jahre
TUD	Lester Kalms
SYNECTIVE	Magnus Peterson

Reviewers

Organization	Participant
NTNU	Magnus Jahre
IOSB	Igor Tchouchenkov Boitumelo Ruf
Sundance	Fatima Kishwar Graeme Parker
Efficient Innovation	Carlota Pons Fabien Marty

Document Description

Deliverable number	D1.2
Deliverable title	Reference Platform v2
Work Package	WP1
Deliverable nature	Report
Dissemination level	Public
Contractual delivery date	2017-10-01
Actual delivery	2018-05-16
Version	1.2.5

	Written by	**Approved by**
Name Signature	Philippe Millet	

Version history

Version	Date	Description
1.0	2018-01-31	Initial version
1.2.5	2018.05.16	Final version

Executive Summary

This deliverable presents the current progress towards defining the reference platform. It is the second version of a document that presents the reference platform. In such it extends the first document going towards the final handbook of the starter kit presented in D1.1.

D1.2 must be seen as the first version of the final D1.3 document, therefore from this version to the final one, some parts will be added and some other parts will be improved and extended. More information will be given e.g. more interfaces or more components will be explained and reviewed.

While D1.2 explains the methodology to select and improve the guidelines, D1.3 will reference them.

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

Image processing deals with image manipulation, transformation and analysis. This domain takes two-dimensional data as an input, whose width, height and pixel depth depend on the sensor. The wide variety of sensors and application types make image processing a complex and deep domain.

The image processing field has already several groups that have been formed with time, all over the world, to deal with sub-domains but no particular focus has been done on low-power. Even though the topic is somehow addressed, it has so far not been the main concern.

While writing about low-power, it must be explained what is meant in this document because any platform might be seen as a low-power platform depending on the domain addressed.

We can also notice that we easily find high-performance targets dedicated to data centres and clouds at the cost of hundreds of watts. At the other end, in the Internet of Things (IoT) realm we can find low-power targets, for less than a watt, but they drastically lack for performance. Yet this is where the "big players" (like Google, Facebook, Amazon, Intel) are today: either high performance computing with higher energy-consumption or ultralow-energy consumption but at the cost of computing performances.

There is nearly a desert in the middle, when one needs a good trade-off between decent-performances and low-enough-power i.e. between 1 watt and 15 watts. We expect this area to develop with the ever growing needs for ADAS systems, but industries with lower volumes (a few thousand pieces a year) might not be able to access the technology. For instance, the Xavier SoC from NVidia is so dedicated to the automotive market and there are so many expectations from the chip manufacturer on this market that it focuses on major customers and sells the SoC only with a strong support for several million euros. The same happens with processors dedicated to smart phones as this market is captive, chips are only sold by millions quantities.

For smaller series of products one needs to find chips that match the processing needs, the price and the power-consumption. When the target has been chosen, image processing engineers still require a good knowledge of what can be done to achieve the best possible trade-off for their algorithm implementation.

The chapter is introducing the goals of the deliverable and explains how to develop an embedded system for image exploitation in real-time as follows: The first step is to get an algorithm that provide a functionality that has to be implemented in a product, the second step is to find trade-offs between the functionality and the physical constraints of the product. When choices have been made, there is

no choice anymore to redesign the platform. This is the kingdom of optimization for everything must fit and run on the platform at the end.

The main objective of the TULIPP project is to identify the best possible choices for embedded image processing platforms and initiate a path towards standardisation. To this end, the work shown in this deliverable is the bases of a book that will help to promote the ideas of the TULIPP project outside the consortium and leave a legacy after the end of the project. It will strongly express that to build solutions for the embedded image processing domain, one needs to fully understand the constraints of low- and medium-volumes (up to thousands of pieces per year) because it strongly limits the freedom of designers.

1.1 Towards the TULIPP handbook

This deliverable is the first step towards a book that will help engineers to implement such image processing systems fast, efficient and saving development costs.

The book will be what is presented in this deliverable: a collection of knowledge from experts of several fields covering the implementation of product using image processing.

The information addresses both experts and beginners. While a beginner will find most of the book useful, an expert should look at the parts that address topics where he is not an expert. E.g. an expert in image processing algorithms will have interest in reading the part on hardware choice and operating system implementation to understand what trade-off he can make on the algorithm to make it easier to implement, while an expert in operating system will most probably like to read the specificities of image processing algorithms to understand what services should be developed in the operating system.

This future book is also aimed at student to learn from the field of embedded image processing which means computing performance with a limited power budget, or better said power-efficient high-performance image processing.

This deliverable is the first extract of the knowledge shared and owned within the TULIPP consortium. We have got agreements and commitment from other projects to cover several chapters of the book to help us extend the number of topics addressed in the book.

The deliverables D1.2 and D1.3 are two versions of the same document. While D1.2 is a draft document collecting a part of the knowledge, D1.3 will be an almost finished version of the contribution of the TULIPP consortium to the final book.

1.2 The Functionality

When starting to develop an application for real-time image processing, one often wonders where to start with. Such an application has indeed complex and contradicting constraints leading to deal with challenging compromises.

Having in mind the services that one wants to deliver, it often starts with a proof of concept application developed on a standard Desktop-PC. The main reason being that then one can concentrate on the algorithmic parts of the application, taking no care of any other constraints like memory size, power consumption and even interfaces for input/output signals. At this step the only concern is to have MatLab or C code giving the expected functional results. This proof of concept shall not even run at real-time, which means it shall not process the input at the speed of a real-life sensor and can afford to be much slower. The input data is often taken from a file and can be generated with environmental simulators or recorded from a real sensor.

This first instance of the code is developed and linked against standard libraries so that the abstraction level is high enough to play with the parameters of the algorithms only, taking no care of any optimization.

This first step must really be seen as a step leading to the specifications of the functionalities. A danger would be to consider this first step as a first implementation of the algorithm on a target. Even though the code resulting from this step runs on a computer, it is far from being the finale platform of the product. The constraints are clearly different and the resulting software will also have to be very different. Developers and managers must clearly understand that the code resulting from this step will have to be rewritten to fit and run on an embedded system.

Some developers are very keen on using various high-level libraries and development techniques known only to experts, requiring many years of training. On one hand, this allows them to be very efficient and faster produce solutions that they can exploit to develop new functionalities and validate their ideas. Those libraries are often quite complex with many functions, many data structures representing hundreds of thousands of lines of codes spread over hundreds of files. Even with open source libraries one has to face the difficulty to reverse engineer them. A wrong idea would be to think that these libraries should be ported to the embedded system to ease the porting of the application.

At this step, the right word would be “to implement” the application on the target rather than “port” the application. Practically, the application must be redesigned to rely on the embedded libraries and operating system. The way to “translate” the application from the functional code to the embedded code depends strongly on the target itself. The software stack implemented on the target requires making choices almost as impactful as choices for the hardware architecture.

A practical approach is to keep the two spaces separate: One time for functionality, one time for implementation on the embedded product.

When the functionalities are satisfying the functional requirements, the main information for the next step is to be able to extract from the specifications of the algorithm the required computing performances (e.g. number of operation per seconds, input and output throughput…), reference stimulation data (sample input data) and reference output data (the results given by the algorithm from the reference stimulation data).

1.3 The Trade-offs

With a clear view of the required computing performances and the product constraints, one can start designing the platform.

The first step here is to get an understanding of all the constraints of the product. Some constraints, like power-consumption, are more obvious for engineers – because it directly impacts the design of the system – than some others, like the price – because it indirectly impacts the design and, for some expensive products, less impactful. Therefore, understanding all the consequences of a choice is crucial. Let's take the example of a chip, which is one of the most important parts of hardware design.

As hardware design teams are dealing with embedded products, they are often limited by the batteries life time and so by the power consumption, but even when the product has access to the electrical grid it is limited by the thermal dissipation of the heat sink, the cabinet and the packaging. Both the power consumption and the thermal dissipation will limit the type of chip that can fit the system. A short way to express this goal could be: the "lighter" the chip, the better.

At the other end, the software design teams have more and more functionalities with more complexity and have to deal with bigger and bigger data sets. There is no other choice for them; everything has to be implemented on the hardware platform. A hardware that would be unable to run the application and provide the functionalities would be of no help and would have to be redesigned. Thus, for software teams, more computing resources means less difficulties to implement the functions, more available memory means easier access to the data and easier scheduling. A short way to express this goal could be: the "heavier" the chip, the better.

Project and product line managers have to deal with customers. They want the product to fit the costs and they need it to be delivered on time. Any delay will cost money and give more time to competitors to develop the same functionalities on their products. Managers have to take costs into accounts. Costs are divided into two categories: the Non-Recurrent Costs (NRC) and the Recurrent Costs (RC).

NRCs are costs involved during the development phase of the product. Thus if one needs more time than ones competitors to develop the same functionalities, the NRC will be higher than the one of the competitors. Higher NRCs might not be a problem when one develops big series of products, but when one develops only few pieces a year, one has to take great care of the NRCs and reduce them as much as possible while it will have a greater impact on the final product price since it has to be divided by the number of pieces that will be produced. A short way to express this goal could be: the easier to program the chip, the better.

RCs are costs involved during the production phase of the product. Thus it strongly depends on the choices made at design time. More expensive components mean higher prices. Higher RCs will always impact the final product price but unlike the NRC one has to take greater care with larger-series and is even critical for mass production where each cent saved on a product will lead to millions in gained in revenue for the company. A short way to express this goal could be: the “cheaper” the chip, the better.

While some of the constraints will lead the hardware designers to prefer domain specific chips delivering more computing per watt with higher power-efficiency; other constraints would lead them to choose more general purpose processors because they are easier to program and will help to deliver faster the product. Some dedicated chips like Field Programmable Gate Arrays (FPGAs) will often offer higher efficiency at the cost of longer developments and higher component prices. The matter is further complicated by system software: A more complex chip with a better operating system and more convenient development tools may result in lower development time than a simpler chip with less mature system software. So the designer must find the trade-offs that allows for implementation of the algorithm with corresponding performance at a given price with the thermal or energy budget that is allowed in the system.

A trade-off does not mean the hardware designer has to find a solution that matches all the constraints. More than often he has to discuss the constraints and move the thresholds. If the functionalities cannot all be implemented; check if it is possible to remove some of them or reduce the effectiveness or the accuracy of the functions within certain margins. Reducing the accuracy from 1% or 2% might in some case reduce the load from several tens of percentage points and allow for using smaller and cheaper chips.

For real-time image processing, the main problem is data throughput and memory size. Images are 2D-data. For colour images, a pixel is often described with a 32-bit word. The total quantity of data to move between memory layers for each function might be tremendous. The processing element is not the only one element to be taken into account while designing the system. The whole hardware architecture must be studied carefully. The memory size, the bus width and the type of IOs are as important as the processing element. Each hardware element will most likely also give constraints on other hardware elements and impact the final architecture choices.

1.4 The Need for Efficiency

The first step to select the hardware platform is to know the targeted price. This selection will be done taking into account the number of products per year. Another primordial factor is the energy budget, the battery life-time and/or the thermal dissipation. With those two figures, price and energy-budget, a first selection of targets can be done. The energy budget is often linked with weight while higher power consumption means more energy storage resulting in heavier batteries. Heavier batteries will also be bigger and take more volume. Bigger volume and weight will require a redesign of the system. If the computer is embedded on a flying drone, the additional volume deteriorates flight characteristics and weight reduces fly time very strong.

The second step is to deal with the needed computation resources. This is not as easy as it seems. Many years ago, we used to look at the clock frequency to compare General Purpose Processor (GPP) performance while we were looking at the number of Multiply-Accumulate (MAC) per seconds for Digital Signal Processors (DSP). DSPs were compared with FPGAs also using MAC/s values while FPGAs were compared one to another according to the number of Look-Up Tables (LUTs), logical cells and DSP blocks (a DSP block is an element inside the FPGA that offers multiply and accumulate capabilities and should not be confused with the DSP processor which is a chip).

With the emergence of multi-core and many-core architectures, the concurrency to access IOs and memory limits the cores and their computing performance as they have to enter idle states to wait for data. The characteristics of the chip have to be tested according to each user application profile for they may use the resources in very different ways. In order to get representative figures and measurements, one must test the chip with benchmarks representative of the final applications. The characterisation of the memory bandwidth and the measurement of the performance of the cores for several basic functions with several data set sizes are a good way to proceed. For instance, one may test Fast Fourier Transforms (FFTs) with several sizes and check how the core performance will be affected by the size of the data set while with this kind of function the distance between data increases with the size of the data set.

Some processors also have accelerators for dedicated basic functions. An example of such an accelerator is the Streaming SIMD Extension 2 (SSE2) implemented on the Intel processors. An SIMD is a set of cores processing a Single Instruction on Multiple Data. It means there is one controller that sends the same instruction to many processing cores. Each core, at each cycle will execute that instruction with data that are different from the one received by other cores. This kind of processor is very energy-efficient as the control is reduced to its minimum, but the efficiency requires that the application can be data-parallelized. There is usually a reduced set of instructions that can be handled by the processing cores. This kind of accelerator must be used when available because they allow using

all the available computing resources which gives the best possible performances. When one would only use the general purpose processor he would consider the chip not suitable for the application while it could actually be if the test were fairly done. Most of the time, the difficulty for using such accelerators comes from the tools. The hardware often offers new functionalities long before they have time to be integrated in the tools. When the SSE2 was just implemented on Intel processors, one had to write assembly code to access the hardware accelerator while today it is taken in charge in compilers. The same happens today with the Myriad processors which efficiency and best performances is only available to well-trained programmers.

Another possibility to access the best performance is to use vendor libraries. General Purpose Graphic Processor Units (GP-GPU) are able to compute almost any kind of function and are extremely parallel. They are designed to efficiently schedule the memory access so that the cores are waiting as little as possible. Doing so, they maximize the utilisation of their cores. One way to implement an FFT on these processors is to program it manually with a dedicated language (CUDA or OpenCL). But the best efficiency is usually achieved with the library delivered together with the component.

Memory size and memory hierarchy is also an important factor to take into account while designing a hardware platform. Since one of the main problem for parallelism is data access, the closer the memory to the core the better. But having memory close to a core does not solve the global problem of data access and data movement. Because the memory size given to a core will never be enough to store the complete set of data given to a function for a given application, it is necessary either to have “big” memory spaces or to be able to process the data as they are streamed out of the sensors.

“Bigger” memory means more possibilities to store the data but at the same time higher latency to access them and also more power consumption. Streaming at the other end will allow for shorter latency and lower power consumption, but might not allow some algorithm to be implemented, e.g. if one needs to compute an average on the complete image to shift the histogram i.e. add or subtract this value to every single pixel of the image. Then, one needs to be able to store the whole image and have no choice but to implement a memory big enough for that purpose.

Depending on the criticality of an application, the underlying platform will be totally different. When, for instance, one needs to be able to master hard real-time deadlines, meaning missing a deadline is not allowed, one will not enable the cache in the processors. Although the processor cache is meant to better feed the processor with data and by this means increases the performance of the hardware, it is also, at the same time, not possible to characterize properly and will result in impossibilities to know the behaviour of the application at design time. This goes also with the number of cores that are used on a symmetrical multi-core chip. The need of knowing the behaviour at design-time and ensuring that this behaviour is the same at run-time, requires composability in the platform. This is not

compatible with current implementations of multi-core processors because cores will have concurrent access to the memory and make the global application behaviour unpredictable.

On the other hand, both the cache mechanism and the multi-core architecture allow higher average performance and will be helpful when only the performance matters. They will also allow for higher energy-efficiency in terms of performance per watt.

So the efficiency one can get from the platform depends highly from the requirements of the application.

1.5 Standardization

Standards are the key to easier platform integration and wider acceptance and utilisation of a solution. There are a lot of examples of widely used interfaces (USB, Ethernet, …), used every day and allowing us to share data and work. However, we might wonder how these standards became so popular.

To spread widely, a standard must fill a need and/or be easy to use. However, it may still fail because of royalty fees. A good example is HDMI. The format is good and quite widely accepted but with one needs to pay to implement an HDMI interface. This is how DisplayPort came up. As it is royalty free, anyone can implement this interface without having to pay. And because of the fees, two standards are now fighting one another making life difficult during meetings when a computer has to be plugged to a screen and converters are needed to interface them. Open standards is a good way to avoid this and it helps to promote and improve a standard because anyone can access it and join the effort of defining. The drawback is to deal with other companies with other constraints to define the standard but the benefits of compatibility and lower prices overrides the decision.

Some companies are voluntarily not using standards in order to prevent anyone to connect anything on the platform. This is for instance the strategy of Apple that wants to keep control of the platform and do not allow for anyone to connect anything to their computers. Thus they design interfaces and plugs that are only compatible with other Apple equipment. However, even Apple uses WLAN, Bluetooth and USB standards as they need to communicate with devices from other companies.

Before starting to envisage the concept of a new standard, one must have a deep look at existing solutions. There might already exist standards that are almost satisfying the needs. If they would not fit completely, they might be adapted to fit better new environments.

For instance, let's have a look at the PCIe standard. There have been several improvements from the first version in 2003 (v1.0) to the very last proposed specifications in 2017 (v5.0) of this standard to cope with new needs, new performances and new applications and to use new capacities allowed by

technology improvements. From v1.0 to v5.0, the throughput has been multiplied by about 15 while improvement was also done to reduce the power consumption during idle periods but not yet to cope with low-power applications. One expected, still in draft extension of the standard is the Mobile PCIe specification (M-PCIe) where power consumption would roughly scale with the amount of data transferred. Rather than creating an ad hoc interconnect that would fit low-energy consumption for embedded devices, one has first to check if existing standards are evolving towards this goal and evaluate if they would match the requirements. This is what the TULIPP project evaluates through this document.

2 THE GUIDELINES:

A path towards standardisation in the low-power image processing system domain

The initial approach was to use the interfaces of the TULIPP platform instance to create a basis for later standardisation efforts. Although this approach is valid and an interesting starting point (see Section 7), we are concerned that it will not result in a universally applicable standard for low- and medium-volume image processing systems. The reason is the heavily constrained design environment of these systems. Image processing applications need to achieve high performance while meeting strict cost, energy, power and physical size constraints (see Section 1.3 for a detailed discussion). A standard that does not have sufficient flexibility to meet these constraints for a wide range of applications is doomed to fail. To reduce the impact of this problem, we came up with the idea of guidelines.

A guideline is an encapsulation of an *advice* and a *recommended implementation method*. The advice captures expert insights in a precise, context-based formulation and orients the follower (the person reading the advice) towards a goal. The recommended implementation method suggests interfaces and steps that work well in practice to implement the advice. The guidelines will help a standardisation body, such as the EMVA, transform the interface-driven description provided by the reference platform into a viable standard. The guidelines provide a basis the standardisation body can use to engage in informed discussion with key stakeholders across the European low- and medium-volume image processing industry. This initiative has already started during the TULIPP project with the setup of an ecosystem of 20+ stakeholders across Europe; most of them are already members of various associations like EMVA. Thus, the guidelines provide a necessary context for adapting the interface-specification of the reference platform into a highly successful European standard.

Both the advice and the recommended implementation methods are supported by theoretical and experimental evidence that is either gathered within the project or is pre-existing in the community. The advice in a guideline is essentially inarguable since it is based on proofs and facts. However, it is not necessary to treat the recommended implementation method as a strict rule. Using alternative implementation methods or new tools is allowed and encouraged.

- ***Advice***: Exploit both vectorization and multithreading for high performance on multicore processors with vector units such as the ARM Cortex A9. On these architectures, utilizing all hardware execution resources is key to achieve high performance [2] [4, 5].
- ***Recommended implementation method***: Use OpenMP. OpenMP is a widely supported parallel programming API that enables programmers to express vectorization and multithreading operations concisely using compiler directives. Programmers need not worry about specifying scheduling and synchronization operations in code. These are handled transparently by the OpenMP runtime system. See the official OpenMP examples[6] to understand in more detail about exploiting vectorization and multithreading simultaneously.

Figure 1: Example of a guideline

Figure 1 contains an example of a guideline for obtaining high performance on multicore processors with vector units.

In the guideline example in Figure 1, the advice orientates the follower towards the goal of achieving high performance in the specific context of multicore processors with vector units. The advice advocates simultaneously exploiting all execution resources for higher performance and provides experimental evidence as support. The recommended implementation method points to OpenMP as a productive choice and links to usage examples.

Guidelines stem from expert insights in image processing and embedded system domains and cover performance, power, and productivity aspects. Generation and evaluation of guidelines is discussed in detail in Sections 2.1 and 2.2.

Guidelines may overlap or exclude one another as a natural consequence of the vast design and implementation spaces for low power image processing. Overlapping guidelines are those whose advice are based on the same underlying principle, or those whose recommended implementation methods are the same. Exclusion occurs when a pair of guidelines point in different but equally competent directions to reach the same goal.

Guidelines are targeted towards both developers and vendors. Developers conform to the guidelines by paying attention to the encapsulated advice and considering recommended implementation methods. Vendors ensure recommended methods are available in their products to attract developers. If implementing recommended methods is infeasible due to constraints, then developers and vendors can consider alternative methods.

We call the process in which a vendor identifies or implements recommended methods compliant with a relevant subset of guidelines as *instantiation,* as shown in Figure 2. The output of instantiation

process is an *instance*. The platform instance is an example of instantiation if the project consortium is considered as a vendor.

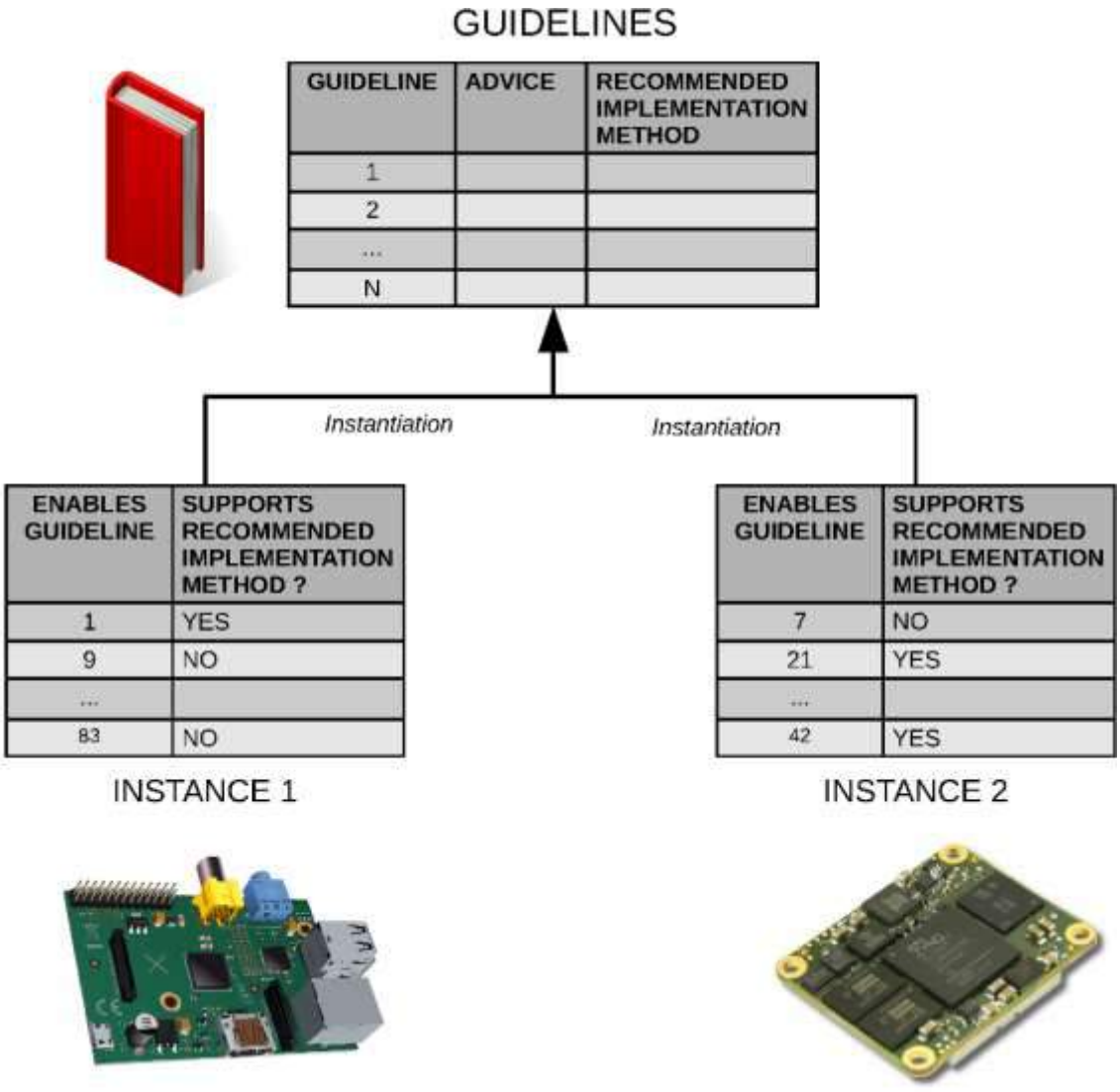

Figure 2: Instantiating platforms based on guidelines. The two instances are partially compliant with different subsets of guidelines.

An instance can be *partially compliant* or *fully compliant* with the guidelines. A partially compliant instance provides alternative methods to implement guidelines, whereas a fully compliant instance provides only recommended implementation methods for the guidelines it supports. Both instances in Figure 2 are partially compliant. The platform instance is an example of a fully compliant instance since it provides only recommended implementation methods for the guidelines it supports.

It is important for vendors to note that an effort to support the complete set of guidelines is not recommended. Support for all guidelines will likely lead to an over-designed instance that is no longer relevant to the application and violates hard constraints such as cost and total power consumption.

2.1 Methodology to create guidelines

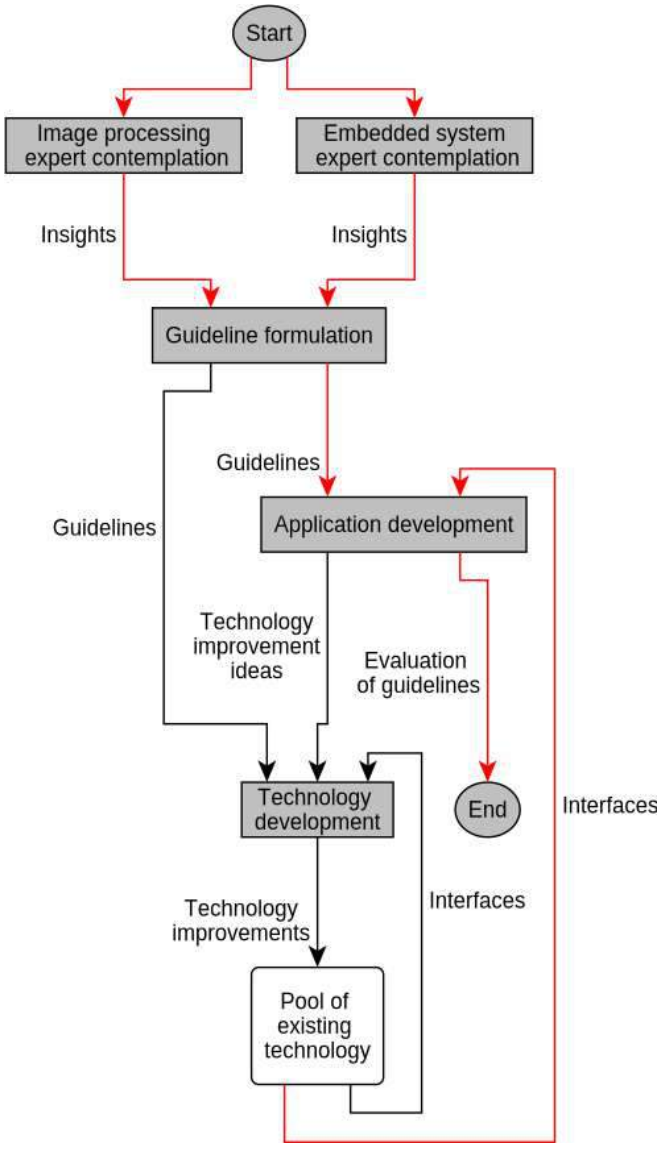

Figure 3: The workflow to generate and evaluate guidelines that define the platform instance. The critical path is shown in red.

Figure 3 shows our guideline production workflow. The requirements of the workflow are that it should (a) generate guidelines for low power image processing, (b) evaluate the guidelines, and (c) be aligned with all project objectives. As evident from the illustration, the workflow captures project-wide effort towards a common vision.

We explain the different steps involved in the workflow starting with the critical path next.

1. Image processing and embedded system domain expertise: The workflow starts by gathering insights from experts in the domains of image processing and embedded systems. Insights refer to deep, underpinning knowledge that come from experience and modelling. The expertise required to produce insights is not restricted to the project partners. External experts such as those in the advisory board can also contribute with insights. Insights have no restrictions on format and can be expressed in any enlightening manner. The key areas of interest are low power, high performance, and high productivity.

> ***Insight**: Real-time OS (RTOS) provide determinism often at the cost of performance.*

As a running example of the workflow, consider the insight expressed by an embedded systems expert as a single line of text:

2. Guideline formulation: Gathered insights are analysed and formulated into guidelines. This involves judging the context and orientation of insights, translating them into advice, and deciding on a recommended implementation method. Translation is necessary since guidelines are goal-oriented, precise, and context-based whereas insights have no restrictions on their format. There is no one-to-one mapping between a guideline and an insight. Many insights can coalesce to produce a guideline, or a single insight can be fertile enough to produce several guidelines.

- ***Advice**: Balance between deterministic execution and performance while setting real-time OS (RTOS) parameters[1].*
- ***Recommended implementation method**: Configure the RTOS using vendor suggested methods[3]. Test and document candidate configurations extensively.*

Continuing the running example, the insight is translated into the following guideline:

3. Application development: In this step, guidelines formulated are implemented within the applications and evaluated for impact. In the context of the project, the applications are the project applications part of the starter kit. Guidelines are implemented either using recommended or alternative methods. We assume that interfaces from existing technology are adequate to implement guidelines, but also recognize that existing interfaces might lack productivity. In the context of the project, technology to implement guidelines is made available through the platform instance. Guidelines are evaluated for goal orientation, the suitability of the recommended implementation method, and arguments for choosing alternative methods. We expect that all formulated guidelines will be evaluated in one or more applications since they have a common driving theme – low power high performance real-time image processing. Once all guidelines are evaluated, the workflow ends.

An excerpt from a possible evaluation of the guideline formulated in the running example could read as follows:

The medical image processing application missed 20/100 deadlines when the default Linux kernel 4.7.2 provided by Xilinx was used. No deadlines were missed under the HIPPEROS RTOS configured with the Least Laxity First scheduling policy, but performance was restricted to 8 FPS, well under the required 24 FPS goal. Enabling multicore execution (SMP) and a hybrid scheduling policy with both data locality and laxity awareness increased performance to 28 FPS with no deadlines missed – a clear win for HIPPEROS. We configured HIPPEROS using the HIPPEROS configuration plugin provided by the project for Xilinx SDSoC 2016.2.

4. Technology development: This is the only step of the workflow that is not on the critical path. In this step, existing technology is improved to enable productive implementation of

guidelines during the application development step. The stimulus to improve existing technology comes from application developers in the form of technology improvement ideas. The improvements aim to increase the productivity of recommended implementation methods to levels that can compete with alternative methods. When recommended implementation methods promise high productivity, developers need not waste time deciding on more productive alternatives. Vendors are more likely to provide recommended implementation methods in their platforms to gain higher compliance. In the context of the project, technology improvements are delivered through the platform instance.

Examples of potential technology improvements include application specific evaluation boards, high performance library routines and custom IP blocks for image processing, low power enabling patches to RTOS schedulers, improved performance analysis techniques for Zynq SoC, workflow tweaks for Xilinx SDSoC etc.

The HIPPEROS configuration plugin provided by the project for Xilinx SDSoC 2016.2 is the technology improvement demonstrated in the running example in the previous step.

The workflow is not iterative since there are no feedback paths or cycles. Feedback is unnecessary since guidelines are based on proofs, facts and expertise. In addition to being feed-forward, the workflow proceeds in a pipelined manner. Guideline formulation starts once a few insights are available. While experts are in the process of extracting new insights and guidelines are being formulated using available insights, application developers continue to build applications to satisfy requirements using available expertise and interfaces from existing technology. Whenever guidelines become available, application developers shift to evaluating them and at the same time benefit from the orientation offered. Technology development starts when application developers realize that recommended implementation methods in guidelines have lower productivity than alternatives and present ideas for improvement.

Since the workflow to generate guidelines is explicit, the set of guidelines defined during the project is not fixed. It can grow continuously by including new guidelines, by evaluating guidelines in new applications, and by making implementation methods more productive. This enables the outcomes of the project to have a long-term and far-ranging impact. We envision that the workflow will be administered beyond the project by the ecosystem of stakeholders created during the project.

As defined through this process, a guideline captures a part of the knowledge corresponding to a very specific topic through a particular experience. Because of this way to capture guidelines, it might not be generic enough at its first stage to be applicable to any development.

Therefore, a methodology was defined to measure the general usefulness of the guidelines.

2.2 Methodology to select guidelines

The process of creating new guidelines is not as easy as it might seem from far. The main difficulty is to issue insightful guidelines that will impact a wide number of developers. While the process for creating guidelines start from particular practice or issues, a more general and global view of the problem as well as a higher level of information is required. We defined the following methodology to derive from the individual experiences gained while developing platforms and applications guidelines with general usefulness meanings.

Four steps have been defined to review and improve the quality of guidelines: (1) each guideline is reviewed and (2) a report is written during this review. If the guideline has enough potential to be added to the guideline database, thanks to the review report, the quality of the guideline is improved (3) and the guideline is added to the guideline database (4).

2.2.1 STEP 1: the Guideline Quality Assurance Board

To assess the quality of the collection of guidelines, the first step is to identify to whom the guidelines is intended - e.g. application developer or hardware designer - while this person is the best qualified one that can assess the insightfulness of the guideline in the utilisation context. Note that there might be more than one person addressed by a guideline and therefore more than one person that will assess the quality of the guideline. We refer to this set of persons as the Guideline Quality Assurance Board.

The work package leader of WP1 is in charge of identifying the type of persons addressed by the guideline and assigns, in collaboration with the partners, the review of the guideline to the best suitable person in the consortium. When the leader of WP1 cannot identify the type of persons addressed by the guideline, the executive board of the project will be assigned the task and will decide to whom the guideline is addressed or if the guideline has to be dropped due to a lack of significance.

Each guideline has a specific quality assurance board composed of experts[1] from at least one of the following groups:

- *HW designers*: dealing with the design of the hardware platform, the components choices and the interfaces according to system requirements.
- *OS designers:* dealing with the operating system design and development facing the need to bring standard APIs for applications and making the OS work on the hardware platform while bringing the application the means to efficiently control the hardware behaviour.

[1] Note that a single person can have expertise in multiple areas and therefore fill different roles in the guideline evaluation board. However, an expert is always given a designated area to focus on when evaluating a guideline to ensure that all aspects of the guideline are fully addressed.

- *Toolchain designers:* dealing with the application developers. They must bring a comprehensive tool that allows the application designer to efficiently map the application on the hardware.

- *Application developers:* knowing the algorithms behind an application and having the ability to develop the code on the chosen hardware. They understand the complete stack and can make use of the tool to develop faster the application.

- *System architect:* dealing with the whole system definition from the definition of the constraints that comes from the final product, having in mind the targeted price of the solution and able to understand integration issues that come with any choice from the other four groups.

- The current list of experts is provided in Appendix 11 – Guidelines evaluation experts.

2.2.2 STEP 2: Quality Assurance report

The assessment will lead to a quality assurance report that will assess the quality of the guidelines.

The report has several goals and will address them all:

- It identifies the type of person to whom the guideline is intended
- It identifies the expertise domains required to write the guideline
- It points out the parts of the guidelines that are e.g. too restrictive or too use-case specific.
- It identifies the parts of the handbook (and also D1.3) that refer to the given guideline
- It identifies the work done in the project that is benefit from the guideline

We rely on the wiki to write the quality assurance report

The outcome of the evaluation is a decision on if the guideline has sufficient quality to become part of the TULIPP guideline database.

2.2.3 STEP 3: The Guideline Improvement Expert Board

If the guideline is not of sufficient quality, the quality assurance board will assign an expert of the topic addressed by the guideline. The expert will be in charge of taking the comments into account and will improve the guideline quality.

There might be more than one domain of expertise and therefore more than one expert required. We refer to this group of expert as the guideline improvement expert board.

When the board is composed of more than one expert, one of them will be assigned as the leader of the guideline evolution and will be responsible for integration of the contributions of others.

When the board is convinced that the guideline has reached sufficient quality, it is provided to the guideline quality assurance board for a new assessment.

2.2.4 STEP 4: The guidelines database and guideline references

When the guideline quality is improved, it is then integrated to the final guidelines database that comes together with the starter kit.

Following the evaluation report, the final handbook but also some of the project deliverables will be updated with references to corresponding guidelines. This applies mainly to D1.3 and the final review report D7.2.

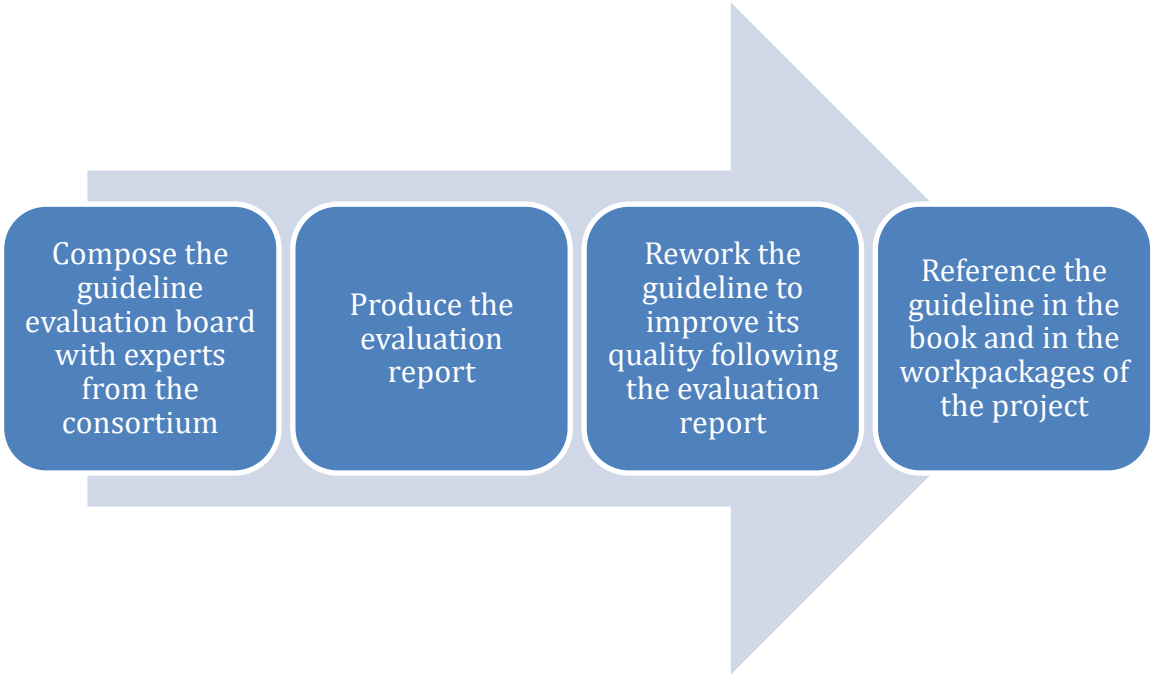

Figure 4: Guidelines quality improvement process

3 EMBEDDED COMPUTING CHALLENGES

Embedded computing refers to a computing solution implemented inside a larger object which primary function is not to compute. The object is often mobile, often has to process data from sensors with real-time computing constraints, and often has no direct, or limited, access to any server or bigger computing infrastructures.

Such embedded systems bring functions like control and automation to devices in common use today like mobile phones, washing machines, cameras, peacemakers, TV, alarm clock, GPS… It is evaluated that 98% of all microprocessors are manufactured as components of embedded systems. [Barr, Michael (1/08/2009) “Real men program in C”. Embedded Systems Design. TechInsights (United Business Media) p.2].

Even though the spectrum of embedded computing solutions is very wide – from a watch to an aircraft computing system – common characteristics and concerns can still be found when comparing typical embedded computers to general-purpose ones. We can notice that embedded solutions are always fighting with small or highly constrained volume, weight constraints and reduced power consumption or heat dissipation. This is referred as SWaP (Size, Weight and Power) constraints.

These constraints have drastic impact on the computer architecture. The whole computing solution being limited, including the processor capabilities, the memories sizes (RAM as well as flash memory). The storage capabilities are also limited; there is often no hard drive. There is almost always links to sensors and actuators. When the device has access to a network, the basic functions of the device can be extended over the network like e.g. the Google GPS application on Android mobile phones using the network servers to compute the road.

3.1 An Image-processing Platform

In the general view, an image-processing platform is composed of a hardware device on top of which low-level software is implemented to operate the hardware (e.g. an operating system and application domain libraries) and a set of tools, generally called a tool-chain, must also be available to develop applications.

Such an image-processing platform is dedicated at processing images out of one or several sensors, a sensor being a camera or camera-like device on most of the applications. Through dedicated algorithm, higher-level of information will be extracted from the image. This result might be used in two ways: (1) to produce actions or (2) to output enhanced information with the image.

Since the sensor outputs frames at a given rate, the platform must be able to process images at the same rate not to lose any information. When the result of the computation is not linked to any safety issue, it might be allowed to lose some of the frames but anyway, for the information to have any meaning, the system must be designed to not lose any frame and thus be real-time. Real-time-processing means the system will process the data at the same pace it is produced.

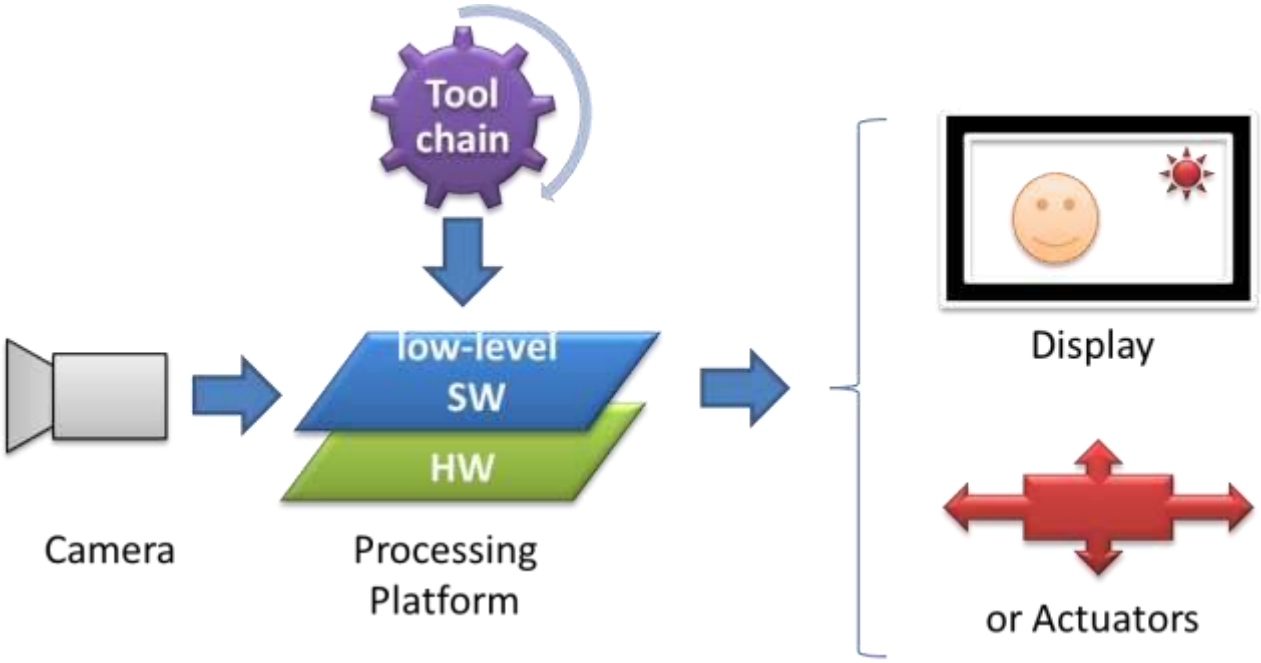

Figure 5: The typical usage for an embedded image processing platform

To achieve this, a holistic view of the system is required so as to achieve the best power efficiency from inevitably highly heterogeneous hardware since the more dedicated the hardware, the better the power efficiency.

This heterogeneity has a cost. While being able to compute more pixels for a given power budget, the programmer loses from genericity and thus programmability and versatility. This means often dedicated programming languages or restricted APIs. Getting the API as close as possible to known and standard APIs eases the learning curve.

When a system gets several computing units, the programmer also has to take into account the scheduling of the tasks on the units and the data transfers between the units. This design step highly benefit from an operating system.

With a power-aware tool chain, the application designer can check, for each mapping of the application tasks on the hardware resources, the impact on power consumption. He or she can thus schedule the

processing chain to optimize both the performance and the required energy. The tool chain would rely on the low-power real-time operating system specifically designed to fit in the small memory sizes of embedded devices. The operating system would come with an optimized implementation of necessary set of common image processing libraries and allows a seamless scheduling of the application on the hardware chips.

3.2 Medical Challenges

As defined by the physicians, Medicine is an art based on science. Doctors have to diagnose, to make prognosis and to take decisions based partly on protocols and scientific examination of the patient. The difficulties they face are mostly to be able to understand what's going wrong with only partial information of a human being. The human body is such a complex system that it requires a lot of practice and experience for doctors to deal with it.

Even if medicine is an art, it is a highly technical domain. With always improving technology, the medical staff can benefit from always more accurate measurement and imagery.

Medical imaging is the visualization of body parts, organs, tissues or cells for clinical diagnosis and preoperative imaging. The global medical image processing market is about $15 billion a year. The imaging techniques used in medical devices include a variety of modern equipment in the field of optical imaging, nuclear imaging, radiology and other image-guided intervention. The radiological method, or x-ray imaging, renders anatomical and physiological images of the human body at a very high spatial and temporal resolution.

Imagery is one of the keys to improve accuracy of diagnosis and reduce the time spent to cure patients. It also allows for faster surgery, smaller cuts in the body and faster patient recovery. All these improvements allow reducing the costs to cure, which is a priority for insurance companies and governments.

X-ray instruments are highly relevant to a significant part of the market share, in particular through the Mobile C-Arm, which is a perfect example of a medical system that improves surgical efficiency. In real time, during an operation, this device displays a view of the inside of a patient's body, allowing the surgeon to make small incisions rather than larger cuts and to target the region with greater accuracy. This leads to faster recovery times and lower risks of hospital-acquired infection. The drawback of this is the radiation dose: 30 times what we receive from our natural surroundings each day. This radiation is received not only by the patient but also by the medical staff, week in, week out.

The fact this device takes pictures continuously, implies x-rays are going through the patient continuously. Since x-rays cannot be narrowed to the zone of interest, the radiation goes not only on

the body part under operation but also to the whole body of the patient and to the medical staff around the patient.

While the x-ray sensor is very sensitive, lowering the emission dose increases the level of noise on the pictures, making it unreadable. This can be corrected with proper processing.

From a regulatory point of view, the radiation that the patient is exposed to must have a specific purpose. Thus, each photon that passes through the patient and is received by the sensor must be delivered to the practitioner; no frame should ever be lost. This brings about the need to manage side by side strong real-time constraints and high-performance computing.

We managed to lower the radiation dose by 75% and restore the original quality of the picture thanks to specific noise reduction algorithms running on high-end PCs. However, this is unfortunately not convenient when size and mobility matter, such as in a confined environment like an operating theatre, crowded with staff and equipment.

Yet by providing the computing power of a standard Intel core-i7 PC in a device the size of a smartphone, TULIPP makes it possible to lower the radiation dose while maintaining picture quality.

3.3 UAV, Drones Challenges

Small unmanned aerial vehicles (UAVs) enter in more and more applications like surveillance and rescue, video production, logistics, research, etc.

The usage of such systems in the entertainment domain is growing especially fast. This development is boosted by the constantly rising commercial market for small UAVs providing broad accessibility, diversity and low costs. Essential enhancements to UAV usage are expected from improvement to their capabilities; perception and intelligent evaluation of the environment make many new applications possible. But more intelligence means more computing power which in turns means more weight and energy consumption. These are however very limited for small UAVs.

As always, each technology comes along with drawbacks and potential for abuse and this is in particular true for UAVs. With the growing number of UAVs, the number of crashes caused by malfunction or maloperation also increases. In the worst case it might cause damages to people, goods and infrastructure. Furthermore, with the broad availability and low cost aspect of UAVs, a common and unforeseeable use (and misuse as well) of this technology is expected in the private sector.

UAVs with automatic collision avoidance will help to reduce those risks.

The UAV use case in TULIPP deals with estimation of depth from two images produced by stereo cameras in order to detect objects and to avoid a collision in further steps. There already exist a lot of algorithms for estimating depth or distance range from different sensors for collision avoidance. In contrast to laser scanners, which are quite robust but expensive, stereo cameras weigh less and are much cheaper. In TULIPP, we focus on user friendly and adjustable depth-map generation which can be used easily for object detection and collision avoidance. While implemented on UAVs, the technology is easily portable for other application on other vehicles and particularly cars.

Obstacle detection is needed to avoid collisions with motionless or moving obstacles on the flight lane and is indispensable for autonomous flight. Most of the small UAVs fly with velocities up to 60 km/h (10 m/s), but some of them can fly with velocities up to 180 km/h (30 m/s) and even more. The stopping distance is strongly dependent from the velocity, the type of the UAV and the payload. With the "usually" velocity of about 30 km/h of a common type of "multicopter", the stopping distance is less than 5 metres, but with more extreme UAVs it can be up to 50 metres. The size of a dangerous obstacle is usually over 5 cm (Figure 6 a), but in some cases obstacles with sizes of as less as 2 mm, more difficult to detect, can also be dangerous (Figure 6 b-c).

a. b. c.

Figure 6: Different obstacles in typical environments of UAVs

The use case solution "collision avoidance" must support the following functions:

1. Real-time pictures collection by two cameras and additionally pictures synchronization of both pictures.
2. Real-time stereo depth map computation and distance measure to object in flight path.
3. Getting the current position and flying route from UAV control unit.

4. Detection of objects which are on the current air route of the UAV and can cause a collision. The relevance of object is dependent on the distance to the object, velocity of the fly and sizes of the object.
5. Calculation of a new safe (free from obstacles) air route taking into account inaccuracies of the computed depth information and possible variations of the UAV from the planned trajectory.
6. Transfer of the new route to the UAV control unit.

All these heterogeneous functions must work in real time, and the TULIPP solution ensures it by its good balance of weight, performance and power consumption.

3.4 ADAS Challenges

Advanced driver-assistance systems (ADAS) while helping the driver to focus on what's important on the road are developing at a very fast pace. Sustained by the now ready, available and good-enough technology, devices are embedding more and more intelligence to analyse the driver and its environment and might even act and control the car in case of danger.

Since this technology saves lives, it is strongly supported by governments and insurance companies for it will not only save people but reduce insurance costs, infrastructure damages and medical care to injured people.

Having more electronic devices in a car has also drawbacks leading to big challenges. The first challenge is the power consumption. With more electronics, more power in drained to feed the chips and this becomes even worth when more computation and 'bigger' processors are required.

A second challenge is the number of sensor that is also increasing at a fast peace. More cameras are going to be implemented to understand the whole environment of the car but also to interpret the behaviour of the passengers and to oversee the driver's actions. Images will also be linked with other sensors in the car and sensor fusion algorithms will be required for the car to get a full understanding of the situation and take the right decision.

This second challenge also comes with a price challenge. The technology is yet developed on high-end cars, but comes fast after only few years to the consumer market. If the target price of the first version is not a problem, its implementation on regular cars must be as cheap as possible. This market is the real final goal while this is where the highest return on Invest is expected.

A third challenge is the ability to foresee situations before they are actually encountered. To this goal, the car must be able to predict the behaviours of the other cars. This can be achieved by communicating cars but, since the traffic is also shared with legacy cars, it must also rely on advanced techniques to analyse the behaviour from what the car sees, just like humans do. Humans do learn

during many years after their driving licence exam to be able to anticipate the behaviours of others. The cars will also have to develop learning capabilities.

All this pushes the technology to develop more image processing chips and algorithms, but also the ability to interpret the image, not only with former processing algorithms, but also with higher level of perception like what is today developed through neural networks.

The demand for more computing power at a reduced power budget, or better said a higher processing efficiency, and cost-effective solutions are crucial for this market to develop.

By 2030, the total price of electronics should reach 50% of the price of the car, compared to about 30% today (source : Statista, Microvision 2016, https://www.slideshare.net/MicroVision/mems-and-sensors-in-automotive-applications-on-the-road-to-autonomous-vehicles-hud-and-adas). The market is improving fast for more automation is requested and this proportion should grow even more within the next 20+ years and the emergence of totally autonomous cars. The sensors will be a set of cameras, some of them checking the road at the front, rear and sides for collision avoidance with other cars, bicycles, pedestrians, etc and for understanding the structure of the road. The car will have to detect the signs and adapt the speed not only to the signs but also to the situation. The cameras will also be used for odometry - i.e. measurement of the position of the car on the track - and to compute the speed of the car. Some cameras will also be implemented in the car to check the driver health and awareness. In addition to the cameras, other sensors might be added, like LIDAR on top of the car to get a 3D understanding of the environment of the car, radar at front and rear of the car to check the distance and relative speed with other cars at relatively far distances, odometry system on the wheels will allow the car to know the distance travelled by the car, ultrasonic sensors will allow the car to compute near distances with obstacles, and GPS will also be used to get the positioning of the car.

The sensors are somehow redundant to achieve the best possible accuracy and allow for measurement in all possible climatic conditions while some sensors might become ineffective with snow or rain. For instance, Electro-Optical (EO) cameras will not give any information with total darkness or with fog.

One of the near challenges is to be able to extract information about the objects around the car from a stream and to be able to extract this information at the same rate as the video. The utilisation of advanced processing like Viola & Jones classifier or Convolutional Neural Networks is one of the most important challenges of the coming 5 years.

3.5 Challenges Conclusion

The challenges are summarised in the following table. More and more processing is required to process more and more sensors with always higher quality, bigger picture sizes. At the same time always higher level of image interpretation is required.

Type	Challenge
Sensors	More sensors to capture the whole environment. More cameras with better quality and bigger image sizes
Algorithms	More complex, requiring more processing from the hardware. More information will be extracted from the images. More intelligence from the images and from other sources of information (other kind of sensors, communications between drones or cars…)
Energy	The energy shall ideally remain constant. While this might not be possible, it must be mastered as more energy means bigger batteries with higher costs and weight. Mastering the power-budget means much higher processing-efficiency is required.
Development Costs	Development costs must be as low as possible and time-to-market as short as possible. To achieve this, the development must rely on standard libraries and APIs. An operating system is required to capitalize on the implementation of optimized power-efficient libraries based on standard APIs.
Customer Price	The markets addressed by TULIPP are highly competitive. Therefore the final cost of the solution must be controlled to be able to offer it at a price customers can afford.

Table 1: Energy-efficient Embedded image processing applications challenges

4 HARDWARE PLATFORMS

Hardware platform refers to the underlying hardware that runs the software. While the software comprises the operating system, the libraries and the final application, the hardware platform is made of the board, the processor(s) and the interfaces both to the outside world (i.e. sensors, display, inputs and outputs in general) and between the components on the board itself.

4.1 Hardware constraints from the system point of view

Being embedded, the hardware part of the system has to fight thoroughly to lower energy consumption and heat dissipation to its minimum. This allows both to reduce the battery and the heat sink size, reducing at the same time the size and the weight of the hardware. This general guideline is necessary and strong, but not sufficient to select each component.

While the hardware has to be integrated in a system with other devices, it must take into account the interfaces of those devices as well as the necessary throughput and computing performances to cope with the application of the final product.

While widely used and made to have some stability in the designs, standards may also evolve and any improvement in any part of the platform should be proposed as a candidate for such an evolution.

4.2 General platform architecture

Anyone has the ability to build any platform but it requires experience to build a platform that fulfils all the needs and requirements. The difficulty leads in the way one selects the components of the platform which at the end have to work all together while providing the expected performance for the given and foreseen development budget.

Added to this, while the products we are focusing on are in an application domain that requires a relatively long lifetime, technology improvement shall be anticipated to select the components relying on interfaces that have the best evolution potential and that will last longer.

To this end, we are reviewing the interfaces and components available today and evaluate them from the point of view of our application domain: low-power image processing with product lifetime of 10 years or more.

This review will lead us to select a reduced number of interfaces that will be part of the platform. Together with the selection we might also discuss some possible improvements that we would benefit from and which could be seen as a proposal for standard evolution.

4.3 Processors

While at the core of the architecture, the processor (which nowadays is more a System on Chip with several to many processors and accelerator cores) has to be chosen carefully.

There are actually two levels or two parts in the evaluation. One part is an evaluation one can do with the datasheet of the component, the other part can only be achieved with proper tests on the component.

From the datasheet, one can collect information that will be a first filter for the chips:

- The thermal design power (TDP) which reveals the maximum watts of heat one has to evacuate off the chip through cooling systems. In the medical use case for instance, the maximal heat that can be dissipated is 8 watts. The board will have to have a global TDP less than 8 watts, which implies the sum of the TDP of the components on the board must be less than 8 watts.
- The operating temperature tells one at what temperature the component will be able to work without any damage to the component. In the ADAS use case, the computer must be able to work when it's frozen outside at -50°C, but also when it's more than 60°C like in deserts. Since this range of temperature is uncommon, it might drastically reduce the number of components.
- The power consumption, when multiplied with time, provides the amount of energy the component needs to work for that given amount of time. This has to be compared with the battery capacity on the product. The battery size might have to be adapted, which will have impact on the global weight of the product. This is an important factor for the UAV use case, while bigger batteries mean more weight. If the solution is too heavy, the UAV might not even take off.
- From the analysis of the programming language and APIs a programmer can understand how easy code can be written for a given chip. However, application development on a given target does not rely only on the way you write code for it, but also on the tools available to deploy, debug, monitor the application execution on the target. In complex hardware architecture, one must also have tools to identify and track deadlocks and bus contentions. Programmers must also have access to a complete documentation, which is too often not available or not written with all the required details of the inner architecture.

- From the subset of processors that can be extracted from this first selection, experiments will be necessary to evaluate further to the application needs:
- Real-time: The pace at which the data is sent from the sensor must be handled by the interconnect to which the chip is connected. The bandwidth, the throughput and the latency of the interconnect have to be taken into consideration, as well as the capability of the chip to deal with DMAs and to manage interrupts. For instance, some processors have very deep pipelines with more than 15 stages that implies latency for every single interrupt.
- Ease of programming: When the development team starts to develop on the target, they can evaluate if it is hard to manage or not.

In the Figure 7 a qualitative comparison of GPU and FPGA is shown [27]:

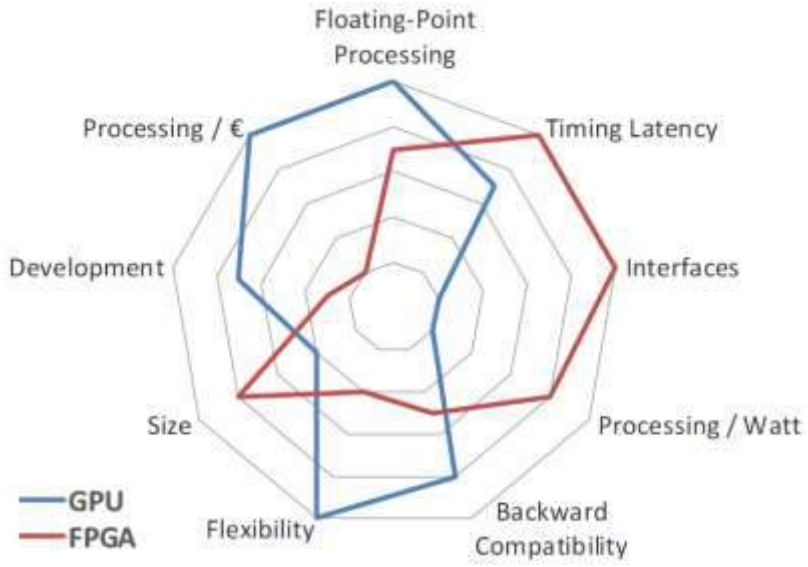

Figure 7: Qualitative comparison of GPU and FPGA

The results of the evaluation on the target will strongly depend on the way it is implemented.

4.4 System on Chip

Chips are more and more complex because they embed more and more heterogeneous components each one being dedicated to a specific kind of processing. Such a system is called a System on Chip (SoC). An SoC embed at the chip level the complexity that we previously had at the board level. The devices implemented on a module typically require a high level of inter-connection, like memories and processor elements.

In D1.3, the following SoCs will be described and discussed, they have been identified in WP2 as good candidates for low-power image processing:

- NVIDIA Tegra X2
- Xilinx Zynq and Zynq MPSOC ultrascale+
- Intel Myriad 2

4.5 Processor Modules

Together with the rising complexity of chip architectures, many versions in the same family are built to cover a maximum of application cases. Thus some versions will get, for instance, video codecs, while some others will get dedicated secured hardware blocks. For SoCs based on FPGAs, several SoCs are available with different sizes of the reconfigurable matrix.

While this allows the user to select the right chip for his application, it might become difficult for a board manufacturer to build as many board types as the number of available chips. Therefore, the solution adopted is often to solder a chip on a smaller board called a module. The module has standardized interfaces. This allows the board manufacturer to develop carrier boards with different flavour of input and output interfaces while keeping the same interfaces with the processing module. Thus one can build his own configuration picking up the carrier board that covers his application needs and install the processing element that copes with the processing requirements.

This is good for evolution as well while it allows an application designer to change the processing element if the application requirement changes and more computation is required.

It is also good for standardisation while it allows the processor to evolve while keeping the same interface with the carrier board. This allows to master the components around the processor and allows to achieve a better stability of the global system while increasing also its compatibility to the global system when we follow the evolution of the chips.

There are several approaches but there is not yet a clear emerging standard. Hereunder we list some of the techniques to assemble such a module and connect it to the main board.

4.5.1 Physical connectors

There is no actual standard to connect such a module to a carrier board. One will actually choose depending on the constraints of the system. In the following we describe some of the most used connectors.

Many manufacturers however use a SODIMM connector which allows connecting the module to the carrier with a limited thickness. For higher mechanical strength, the module can be screwed on the carrier.

Connectors like Samtec QSH-060-01-H-D-A, which is used by NVidia for its Jetson module, have a better mechanical strength than SODIMM because it will be located under the board. While SODIMM will be limited to the size of the edge of the board, this connector will allow having several lines under the board which allows also more wires between the module and the carrier.

As miniaturisation goes on, more and more modules are shipped with fine pitch board-to-board connectors that are optimized interconnect solutions for smaller and thinner electronic consumer products. They are located under the module but are much smaller than the connector used by NVidia.

Except for the form factor, the mechanical constraints and the power supported by the connector, there is no consideration in term of power efficiency that can help the hardware designer to choose a connector among others.

4.5.2 System on Module (SoM)

A System on Module is a printed circuit board that integrate the functions of a whole system in a single board.

A typical use for SoM is to get components that need a high level of interconnectivity on a separate module. It also allows chips manufacturer to select the other components to produce a consistent and mastered system that they know will work. One particular example is processors that only tolerate certain type of DDR memory chips to operate properly.

A SoM is often connected to a carrier board that might integrate several SoMs to form a system with higher complexity.

Even though there are several ways to connect SoM (like physical connector and protocols), to carrier boards and it is difficult to select a single standard that would fit them all; since SoM allows for more mastery of the system and better stability it also reduces the global design costs of the hardware solution and is therefore an advisable way to design a hardware platform.

4.5.3 Embedded System Module (ESM)

An ESM module typically includes a CPU processor, memory, module-specific I/O interfaces and a number of basic front I/O connectors. It is aimed at being plugged to a carrier board.

It relies on the standard PCI bus for the board to board interface. ESMs are typically used on boards for CompactPCI and VMEbus as well as single-board computers for embedded applications. A company standard by MEN Micro, a manufacturer of embedded computers, specifies the ESM concept and the different types of modules. The ESM specification defines one form factor for the printed circuit board: 149 × 71 mm (5.9 × 2.8 in).

4.5.4 PC/104

PC/104 (or PC104) is a family of embedded computer standards which define both form factors and computer buses. PC/104 is intended for specialized environments where a small, rugged computer system is required. The standard is modular, and allows consumers to stack together boards from a variety of COTS manufacturers to produce a customized embedded system.

It currently relies on PCI and more recently (2008) on PCI-express buses.

4.6 Input / Output Interfaces

Depending on the application domain, the hardware has to interface with different kind of sensors and actuators. For integration grounds, the hardware designer may not have choices but to use the already existing interfaces and standards. For instance, the automotive domain uses the AUTOSAR standard that defines interfaces and protocols and we can only advise to use it for any device dedicated to this market. For other domain there are less constraints, therefore more flexibility is allowed.

TULIPP platform uses open standard interfaces where it is possible.

4.6.1 Camera interfaces

In TULIPP systems, each sensor produces 2D-data, but interfaces, protocols, formats and amount of data can be very different. In mostly use cases sensor is a digital camera. Because of that, the following protocols will be described and discussed in D1.3

- GigE Vision
- Camera Parallel Interface (CPI)

4.6.2 Display interfaces

Many TULIPP systems must visualise the processed video on different devices. The following protocols suitable for that will be described and discussed in D1.3

- HDMI

- Display Serial Interface (DSI)

4.6.3 Hardware interconnect Protocols

Hardware interconnect protocols are needed for connection of different processing boards, external components (like hard drive or SSD) and actors (like autopilot of UAVs). Standard GigE protocol can be also used for output in many cases. The following protocols will be described and discussed in D1.3:

- PCIe
- USB

4.7 Scalability of the compute architecture

Scalability can be implemented on many ways:

- Usage of different components the same type (e.g. Ultrascale+)
- Usage of different component types
- Connection of many boards

This part will be extended in D1.3 and will explain how scalability must be taken into account in the hardware architecture.

5 OPERATING SYSTEM & LIBRARIES

When a developer has to deal with an embedded computer, one of the first challenges he faces is the size of the memory and the available computing capabilities.

This often leads the software developer to choose not to implement any operating system on top of the hardware and directly programme the resources manually.

While this is probably a good choice in former times, it is however highly time consuming because each developer has to develop his own set of libraries, drivers and APIs and at the end reinvent the wheel at every project.

Using an operating system allows for a better approach, more structured relying on standard APIs which allows for faster development but also allows to capitalize on the work done on each project to improve the operating system.

Such an operating system must, however, remain as small as possible so that it does not use all the available resources of the system both memory wise and computing-power wise. Having an operating system fine-tuned for an application domain delivering the right and necessary libraries and APIs is the target that we have in the TULIPP project.

In D1.3 the following sections will be developed.

5.1 Low memory footprint

Since the application domain is embedded, the quantity of memory is always a limiting factor. While there is a limitation on the maximum amount of memory the selected processor can manage, it is however mainly constrained because the memory itself leads two other parameters of the computing solution: (1) the power consumption and (2) the price of the hardware.

Whatever the reason, the total memory is limited and the software must take this constraint into account. Since the operating system is at the very heart of the software, it must leave enough space for the application binary code and the data.

5.2 Scheduling policies

The use cases of the project are dealing with real-time. This constraint has to be taken into account by the operating system that must allow the application to react to external stimuli as fast as possible.

Dedicated interrupt management system to deal with the communications (i.e. the input frames and the output buffers) must be implemented to allow such a real-time behaviour.

5.3 Managing hard real-time constraints

Moreover, the medical use case must not only react at real-time but must ensure no frame loss. Specific mechanisms must be implemented in the operating system to warn the application if the system is close to not being able to meet all the application deadline and specific scheduling policies must be implemented in the operating system to describe those deadlines.

5.4 Choosing application programming interfaces

The developers of an operating system for embedded applications must keep in mind to reduce the memory footprint of their libraries, select the right functions to be implemented and implement only the functionalities that are required.

6 STRUCTURED PERFORMANCE ANALYSIS: MEETING EMBEDDED IMAGE PROCESSING APPLICATION REQUIREMENTS WITH HIGH PRODUCTIVITY

The main objective of tool-supported structured performance analysis is to substantially reduce the effort required to implement an embedded image processing solution on a heterogeneous hardware platform. We use the term performance in a broad sense to cover key metrics such as runtime, energy dissipation or power consumption. For image processing systems, requirements are often specified in terms of target frame rates or the maximum acceptable latency from a frame arrives until processing is complete.

Development of an embedded image processing application is heavily tied to the hardware platform it will be deployed on. The reason is that image processing applications often have stringent power and performance requirements. Meeting these requirements commonly require that the application is *specialized* by using accelerators (Borkar and Chien 2011). This results in hardware platforms for image processing being *heterogeneous* by containing a collection of compute units with different performance and power consumption characteristics.

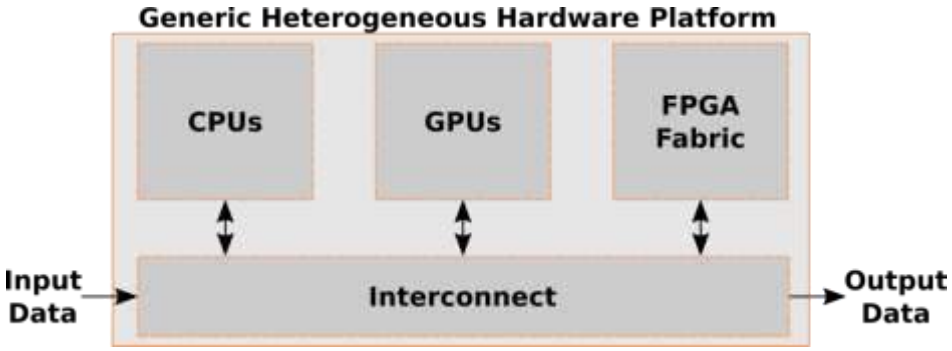

Figure 8: Generic Heterogeneous Hardware Platform

The above figure shows a *Generic Heterogeneous Hardware Platform (GHHP)*. It contains a collection of computing substrates (i.e., CPUs, GPUs and an FPGA fabric) as well as an interconnection network and input/output devices. When discussing performance analysis tools in general, we envision that the target is the GHHP. When focusing on a particular project, we use the concept of a *platform instance* (Jahre et al. 2017). The platform instance is the hardware platform, OS and tools in use in the given project.

The best-practise procedure for implementing an image processing application which meets all requirements is to carry out an iterative trial-and-evaluation process by changing the code and evaluating the effect of the change. After many iterations, the application meets its performance requirements. If for some reasons the requirements cannot be met, even after all possible efforts, then deep modifications, both in the requirements and the solution design, must be redefined. Tool-supported structured performance analysis improves developer productivity by reducing the number of iterations of this trial-and-evaluation loop.

Performance analysis tools are not the only tools necessary for supporting the full project life-cycle. In addition, tools are necessary for instance support regression tests, simulation, version control, configuration handling and bug tracking. Our main finding is that the existing state-of-the-art tools include decent support for such processes and provide options to embed third-party mechanisms for missing features. Therefore, we will focus on performance analysis tools in this chapter.

6.1 The generic development process

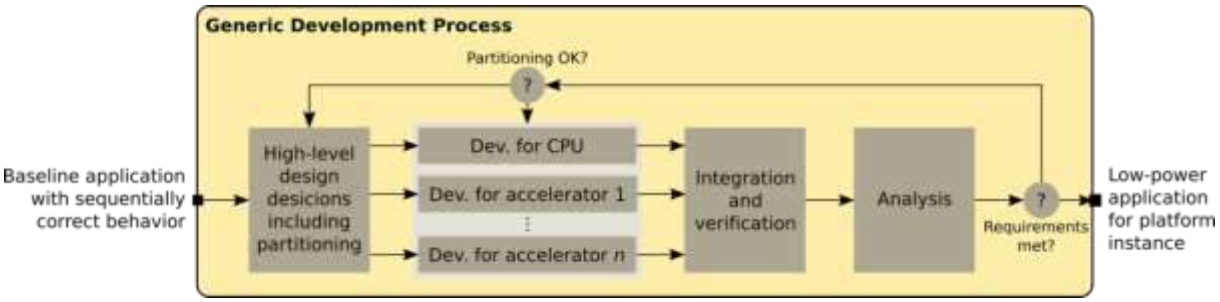

Figure 9: Generic Development process

The above figure shows the *Generic Development Process (GDP)* (Jahre et al. 2017). GDP is an abstraction that captures the trial-evaluation development loop and connects it to all components of a platform instance. GDP is an iterative process for programmers to implement image processing applications that meet low-power requirements while leveraging the heterogeneous processing resources available on the platform instance for performance.

The starting point of the generic development process is the baseline application that executes with correct sequential behaviour on a modern machine with a general-purpose processor. High-level partitioning decisions decide which baseline functions should be accelerated and how. Partitioning splits off into accelerator-specific development stages that later join to produce an integrated application with the same correct behaviour as the baseline. The performance of the integrated application is checked against requirements. If found lacking, the partitioning and development stages

are restarted. In this manner, programmers iteratively refine the baseline application to approach the required power consumption and performance.

In the most basic case, GDP can be carried out manually without any tool support. This will result in very low developer productivity since GDP is reduced to a time-consuming trial-and-error process. The main task of the performance analysis tools is to capture information about efficiency-bottlenecks in the current implementation and clearly communicate this information to the developer. The developer will then be able to make informed decisions on how the implementation needs to be changed to meet performance requirements.

At a high level, the performance analysis tools need to capture two classes of information to help the developer identify performance problems:

- **Inter-compute unit efficiency:** Mapping different parts of the application to the different compute units available in the hardware platform is necessary to fully leverage the capabilities of a heterogeneous platform, and performance analysis tools need to provide feedback on the quality of this mapping. Identification of bottleneck compute units is especially important. Common techniques for achieving this is to profile the application on each compute unit and present an aggregated profile to the developer. The profile needs to include both runtimes/latency and energy/power.
- **Intra-compute unit efficiency:** A performance bottleneck may also be due to an inefficient implementation within a single compute unit. In this case, the developer needs to map the performance problem to the source code responsible for creating it. To achieve this, we need performance analysis tools that can pinpoint performance problems and automatically relate these to source code constructs. This capability also needs to cover both runtime/latency and energy/power.

When the performance problem has been identified, the next step is to change the implementation to remove the problem or reduce its impact. One option is to let the developer address the problems without support. This can be an efficient strategy when the developer can leverage application-specific insights. Performance tools can also help by either implementing fully automatic solutions or suggesting problem solving strategies that have been used successfully on similar problems in the past (e.g. suggest using an optimised library function). Tools for automatic performance optimization on heterogeneous platforms are a challenging research problem that is currently being targeted by many research groups (Bacon, Rabbah, and Shukla 2013).

6.2 Selecting performance analysis tools

Executing the *Generic Development Process (GDP)* can be very time-consuming when application requirements approach the computing capabilities of the platform. Embedded image processing applications tend to be compute-intensive and require installation in systems where power, energy and physical size are first-order constraints. Thus, GDP needs to be supported by performance analysis tools to achieve high developer productivity. In this section, we introduce the state-of-the-art programming models for heterogeneous systems and discuss the key features needed for performance tools to efficiently support GDP in the context of embedded image processing systems development.

6.3 Programming model selection

GDP does not put any restriction on which programming model is used to implement the image processing applications. An important design decision is whether to use a single programming model for the complete application or to accept that different parts of the system are implemented with different programming models. We will refer to these strategies as a *Single Programming Model (SPM)*-strategy and a *Multiple Programming Model (MPM)*-strategy, respectively.

	SPM-strategy	MPM-strategy
Advantages	• Setup is simpler since a single tool chain can be used for the complete applications. • Application maintenance is simplified due to a single code-base and single tool-chain. • The abstractions employed to support multiple, different computing units tend to result in less code being necessary to implement the application.	• Using specialised vendor tools for each component reduces the risk of introducing performance-limiting abstractions. • Platform selection is simplified since there is no requirement that all vendors support the same programming model.
Challenges	• All platform components need to support the chosen programming model. This may limit hardware platform options. • The higher level of abstraction may limit the achievable performance and energy efficiency.	• Application maintenance is complicated by multiple tool chains, especially due to upgrades. • Development and maintenance is more difficult since the company needs to recruit and retain people

		that are experts in each programming model. ● Efficient communication mechanisms and interfaces between the parts of the application that are realised in different programming models needs to be designed, implemented and verified.

Figure 10: Advantages and Challenges of the SPM and MPM strategies

The above table outlines the advantages and challenges of the SPM and MPM strategies. Overall, the SPM-strategy simplifies the development process compared to the MPM-strategy. However, the SPM-strategy may limit the attainable performance and energy efficiency due to a higher abstraction level. In addition, the SPM-strategy complicates platform selection since the preferred programming model needs to be efficiently supported on all platform components. Deciding which strategy to follow is complex trade-off that depends on application requirements, hardware platform requirements as well as the expertise and strategic focus of the company.

With an SPM-strategy, there are two main options:

- **OpenCL:** OpenCL (Stone, Gohara, and Shi 2010) is a standard API that enables program execution on a heterogeneous system which contains hardware components such as CPUs, GPUs and other accelerators. It provides an abstraction layer where each computational device (e.g. a CPU or GPU) is composed of one or more compute units (e.g. processor cores). These units are again subdivided into Single-Instruction Multiple-Data (SIMD) processing elements. The task of the developer is to formulate the program in a data- or task-parallel manner to use the computational resources available in the platform. Although OpenCL guarantees that a program will run correctly on all OpenCL-supported platforms, platform-specific optimization is commonly necessary to achieve high performance and energy efficiency. Although FPGA vendors support OpenCL on selected FPGA platforms, it can be challenging to determine the root cause of performance problems since the OpenCL computing system model does not map clearly to the FPGA substrate (Wang et al. 2016).
- **High-Level Synthesis (HLS):** The abstractions of OpenCL may limit implementation flexibility on hardware platforms that contain reconfigurable fabrics such as FPGAs. An alternative approach is *High-Level Synthesis (HLS)* where the application is implemented in a high-level language such as C. The HLS-tool automatically generates an accelerator for a selected code segment (e.g., a procedure). HLS is a viable design alternative due to the existence of multiple commercial and academic tools (e.g. Xilinx Vivado HLS and LegUP (Canis et al. 2013)). There are two important

challenges when using HLS. First, the tools only support a subset of the high-level language which commonly means that the code needs to be modified to enable HLS. Second, the relationship between the high-level code formulation and the generated hardware is not always obvious which complicates performance analysis.

With an MPM-strategy, the programming models tend to be tailored to the capabilities of each computational device:

- **CPU:** Current hardware platforms for image processing applications tend to contain multiple CPUs. Programming models for multi-cores has been studied extensively, and powerful tools such as OpenMP are available to efficiently parallelize an application using task-based or data-parallel strategies. An added benefit of using tools such as OpenMP is the existence of advanced performance analysis strategies (Muddukrishna et al. 2016; Langdal, Jahre, and Muddukrishna 2017).
- **GPU:** GPUs are commonly part of hardware image processing platforms due to their high efficiency on graphics and image processing tasks. GPU vendors tend to support using their hardware for general purpose computations for specific programming models. For instance, NVIDIA focuses on CUDA while AMD and ARM support OpenCL.
- **FPGA:** Development for FPGAs has traditionally used Register Transfer Level (RTL) models such as VHDL or Verilog. These models require specifying low-level implementation details such as the width of busses, etc. Thus, development using VHDL or Verilog tends to be time-consuming. An alternative approach is to use high-productivity RTL programming models such as Chisel (Bachrach et al. 2012). These models improve productivity by abstracting away implementation details and providing powerful, reusable constructs. In contrast to HLS-tools, the developer still specifies the concrete structure of the hardware.

6.4 Evaluating performance analysis tools

The selection of programming model and hardware platform will determine the attainable performance and energy efficiency of the embedded image processing application, while the capabilities of the performance analysis tools that are available for the chosen programming model and platform will determine how productively an application that meets requirements can be developed. In other words, the existence of efficient performance analysis tools is a secondary concern. There is little point in quickly developing a solution with a programming model that cannot meet performance and energy efficiency requirements.

The existence of advanced performance analysis tools can only impact programming model and hardware platform selection when there are multiple options that can meet requirements. In this case, the performance analysis tools can be evaluated on their ability to:

- Efficiently detect performance problems
- Relate the performance problem to source code construct that caused it
- Provide suggestions or solutions to how the performance problem can be alleviated

Efficient performance problem detection tends to require some form of application profiling combined with high-level visualizations such as Gantt charts or Grain Graphs (Muddukrishna et al. 2016). With appropriate mechanisms, the visualizations can automatically zoom in on problematic sections and thereby significantly simplify performance problem detection.

By leveraging the debug information available in the application binary, it is possible to map a performance problem to a specific source code location. By externally sampling the CPU program counter, it is possible to implement a similar strategy to relate instantaneous power measurements to source code constructs (Jahre et al. 2017).

Providing analysis functions that can automatically solve performance problems is a challenging research problem, and solving problems tends to be the responsibility of the application developer. A different approach is restricting the formulation of programs such that performance problems are less likely to occur (e.g. (Koeplinger et al. 2016; Prabhakar et al. 2016)). Another class of approaches can avoid some platform-specific performance issues by conducting an extensive Design Space Exploration (DSE) to ensure that implementation details are chosen to arrive at a high-performance design point (e.g. (Zhong et al. 2017)). An interesting compromise is to explore semi-automatic approaches where a tool provides suggestions on how a performance problem can be dealt with and the developer leverages domain knowledge to choose the exact strategy.

6.5 STHEM: The TULIPP performance analysis tools

The previous sections discussed embedded image processing application development and analysis in a general sense. In this section, we will give a concrete tool-chain example by introducing the collection of performance analysis tools that are used in the TULIPP project. Overall, TULIPP follows an HLS-based SPM-strategy and use a combination of state-of-the-art industrial tools and novel research-based tools.

The concept of a *platform instance* was introduced in the start of this chapter. A platform instance can be created using any combination of hardware, RTOS, and development tools. However, support for the generic development process in each platform instance is unlikely to be readily available. For example, all the components of the TULIPP reference platform have independent workflows that

partially overlap the generic development process, and at a more basic level have poor to non-existent support for each other. To resolve these incompatibilities, we build utilities to fill the gaps between existing tools and support for the generic development process within the TULIPP project.

We collectively refer to the utilities developed during the TULIPP project as the *Supporting uTilities for Heterogeneous EMbedded image processing platforms (STHEM)* (Sadek et al. 2018). STHEM is designed to be as vendor-independent as possible to simplify implementation for arbitrary platform instances. STHEM includes connecting glue that interfaces independent components together and standalone tools that extend individual components to provide complementary features.

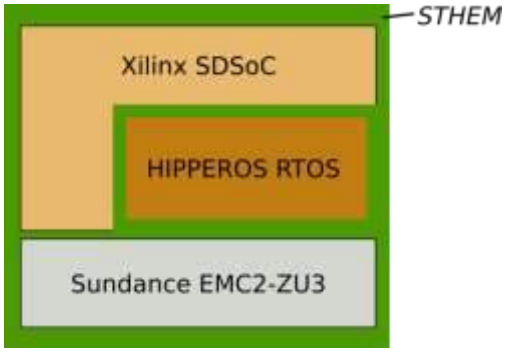

Figure 11: The Supporting utilities for Heterogeneous Embedded image processing platforms (STHEM)

The TULIPP toolchain is a combination of STHEM and existing components of a given platform instance that work together to simplify the generic development process for programmers. The TULIPP-PI1, which is shown in the above figure, is the platform instance that the TULIPP consortium is focusing most attention on. The reason for the attention is familiarity with the components that make up the platform instance. TULIPP-PI1 consists of the Sundance EMC2-ZU3 carrier board with the Xilinx Zynq UltraScale+ MPSoC processor arranged in a two-board configuration to expose a high degree of parallelism to applications. The hardware is operated seamlessly by the HIPPEROS RTOS. Application development tools in the platform instance are custom adaptations of Xilinx SDSoC and HIPPEROS tools to support multi-board acceleration and real-time requirements.

Utility	EMC2-ZU3	HIPPEROS	SDSoC
Power Measurement Utility	No power measurement hardware	Does not quantify task power consumption	Cannot correlate power consumption with application phases
HIPPEROS SDSoC Compatibility Layer		Cannot accelerate tasks on FPGA	No support for HIPPEROS

HW/SW Image Processing Library		Few optimized image processing functions
Design Space Exploration Utility (analysis module)	Cannot auto-tune RTOS and task parameters	Cannot auto-tune accelerated function parameters
Dynamic Reconfiguration Utility	No support for dynamic FPGA reconfiguration	Involved method for dynamic reconfiguration

Table 2: Limitations of the main TULIPP-PI1 components and needed utilities to alleviate the limitations

The above table lists the limitations of the main TULIPP-PI1 components and which utilities are needed to alleviate the limitations. STHEM version 1 includes the three utilities that are necessary to provide a minimal end-to-end image processing system for the TULIPP-PI1 platform:

- **The HIPPEROS SDSoC Compatibility Layer (HSCL):** The HSCL is needed to provide HIPPEROS support in SDSoC, and thereby enabling building HIPPEROS applications with HW acceleration. The utility consists of the configuration files, libraries, and script files needed for SDSoC integration. The HSCL is a glue layer between HIPPEROS and SDSoC. It is not a standalone STHEM utility and is by necessity tightly coupled to SDSoC and TULIPP-PI1. By default, SDSoC supports only FreeRTOS and Linux OS. When the user chooses the OS for an application, the application will be built for that OS, and the OS will be packaged together with the application into an SD-card image. SDSoC is limited in its extensibility to other systems. There is no officially recommended method for a third party to add support for an unsupported OS such as HIPPEROS. This is where the HSCL is needed.
- **The Power Measurement Utility (PMU):** The PMU quantifies instantaneous power consumption of different hardware components and correlates power measurements with application phases. This enables programmers to isolate power problems and make informed optimization decisions. The PMU is designed to be as vendor independent as possible. It only requires access to power supply lines of the platform instance and provides power measurements in a standard trace format that can be viewed on independent visualizations. Vendor-specific functionality is used only to identify application phases while correlating with power measurements.
- **HW/SW Image Processing Libraries (IPL):** The IPL helps programmers implement accelerated image processing applications. As the name suggests, there are two library variants – hardware and software. The hardware library includes IP for I/O devices that handle external image processing (for example, cameras) and image transmission (for example, HDMI). The software library provides optimized software functions for image processing. In the first 18 months of the

project, we have focused on developing the software library with optimized, streaming-capable image processing functions for Xilinx SDSoC. The development of the I/O IP has been postponed until the requirements of the use case applications are fully understood.

We are currently focusing on developing STHEM version 2 which will add support for the following utilities:

- **Analysis and Design Space Exploration Utility:** This utility belongs to the final phase of the generic development process, where the application is working and accelerated on HW, but where the optimal HW acceleration and RTOS parameters have not yet been found. DSE utility will automatically test different variants of the application and estimate or measure performance, power, and logic area for all variants. The pareto front of these experiments will be presented to the developer which can then choose the solution most suited for the application. Parameters explored include pipelining of HW functions, memory usage in HW functions, HW/SW interface selection, and RTOS scheduling.
- **Dynamic Reconfiguration Utility:** Designing platforms enabled for dynamic reconfiguration is an involved process, even for experienced designers. There are many interleaving steps that require deep knowledge of the underlying FPGA architecture. In the next version of STHEM, we will contribute with a utility for Dynamic Partial Reconfiguration (DPR) that will guide the designer in creating applications that use HW acceleration that are configured dynamically in the FPGA. Dynamically reconfiguring FPGAs further reduces power consumption of the hardware platform by dynamically and adaptively exchanging hardware accelerated functions, using only a fraction of the total available resources. The hardware platform can also be modified to use a smaller device to save costs.

  We refer the reader to deliverable D4.1 for a detailed description of STHEM (Jahre et al. 2017).

6.6 Tool-chain Standards

For D1.3, we will compile a list of standards we use within the STHEM tools and as well as the interfaces that are available within vendor tool-chains.

7 PLATFORM, STANDARDS and STANDARDISATION

The aim of this chapter is to collect all the interfaces (HW+SW) used in the book, list them and discuss their usefulness and applicability.

It also explains how the interfaces could be improved to better serve the purpose of high energy-efficient platforms for real-time image processing.

7.1 The hardware platform choice

Within the framework of a project, the development of only one hardware instance can be developed because on top of the chosen hardware the operating system must be developed and the tool chain must work together with the vendor tools.

Considering the constraints of the use cases and the trade-offs that have to be made, the Zynq MPSoC was chosen for it is the most versatile and brings the best energy efficiency for it allows to implement some of the functions of the application to be implemented as hardware accelerators and helps saving power by switching off the unused parts of the SoC.

Since the use cases have different input/output interfaces requirements, we chose to develop a module on a carrier board. This allows not only to adapt the carrier board to the IOs of the application but also to select other chips for future and other instances of the processor module.

PC104 has been chosen because it fits well when rugged computers are required like in cars and UAVs. Moreover, thanks to PC104, the architecture can easily be extended by stacking the boards which is also an interesting feature for automotive and medical applications.

The current PC104 connector might certainly be optimized to reduce its size and be more easily used in very small environments like on some small UAVs and medical devices.

This section will be extended on D1.3

7.2 The operating system implementation choices

Because the application domain is image processing, we chose to implement together with the operating system some of the common libraries used by most of the implementers of that domain. OpenCV has become a de facto standard for computer vision application and offers a set of libraries to efficiently program such applications. While it is largely spread, it is however not possible to compile

the library and deploy it on an embedded architecture while its size exceed the memory of most of the available targets. Therefore a selection has been made in the API of OpenCV to focus on the most useful functions.

This section will be extended on D1.3

7.3 The toolchain choices

The main constraint for the toolchain is the interaction with other components. The toolchain has to interface with vendor toolchains on one side, to the operating system on another side. The operating system interfaces the toolchain with the hardware through drivers that allow the toolchain to access specific counters and probes.

While the interactions might appear clearly, the interfaces are much harder to define.

This section will be extended on D1.3 and a description of these interfaces will be described together with the selection process that has been used.

8 CONCLUSION

This deliverable presented the process to guide designers towards implementing energy efficient image processing platforms.

The currently known interfaces have been presented and we discussed how a selection was made for the TULIPP project and its three use cases.

While not defining a standard, the collection of interfaces and the discussion on possible evolution to improve them can be used to provide the community with better interfaces and higher energy efficiency.

Through the ecosystem and links with other research project, the TULIPP consortium works actively to promote the ideas developed in the project and capture interesting ideas and knowledge from other projects. Together with other projects, we believe that this work is valuable and needs to be transmitted. We therefore started to define the structure of a book that will collect the knowledge developed in this joint effort of providing others with higher levels of energy efficiency for embedded image processing applications.

9 APPENDIX: THE GUIDELINES DATABASE

This chapter contains the list of guidelines produced during the project. The current list is added in D1.2 but will be revised in D1.3 as we will improve their quality through the process described in D1.2.

9.1 Adjust the Algorithm to the Underlying Architecture

Guideline advice

Adjust the algorithm to the underlining architecture. Not all algorithms are out of the box suitable for all kinds of accelerators (e.g. FPGA or GPU). Algorithms that allow stream processing are more appropriate for FPGAs, while algorithms that require random access to memory should preferably be ported onto a GPU. Find a way to optimize and tailor the algorithm to meet the strengths of the chosen architecture.

Audience of the guideline

Application developers; System architects

Author and guideline responsible

Igor Tchouchenkov (Fraunhofer)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Algorithm Architecture Adequation

Insights that led to the guideline

Illustration of the concentric optimization paths of the semi-global-matching algorithm

Initially the algorithm for semi-global-matching (SGM) optimization [1], which is used for the stereo processing in the UAV use-case, was designed for random access of the image data. As depicted in the above figure the algorithm optimizes the result for each pixel along numerous concentric paths. This requires multiple, unordered access to image data and is thus not straight away suitable for the streaming processing of FPGAs.

References

H. Hirschmueller, Accurate and efficient stereo processing by semi-global matching and mutual information, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 2005.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

A literature survey on adaptations of the SGM algorithm to different architectures yield helpful insights for implementation of the algorithm for the UAV use-case. An extensive list of the survey is given in Appendix A. Note that a reduction of the algorithms complexity, in order to adjust it to the underlining Hardware, might reduce its accuracy.

FPGA

FPGAs yield best performance per power.

In order to utilize the streaming characteristics of FPGAs, the aggregation paths should be restricted to the already processed pixels, i.e. 4-5 paths (upper left, upper, upper right, left and current pixel). This reduces complexity and accuracy of the algorithm, but increases computational speed significantly.

FPGAs have limited (fast) memory compared to CPU and GPGPU. Take care of memory consumption!

Power consumption of FPGAs depend on used frame rate and clock frequency. This allows meeting specific requirements, e.g. less power consumption with reduced temporal resolution, without changing the algorithm.

8 path aggregation vs. 4 path aggregation

GPU

GPU allows computing the SGM algorithm massively parallelized without larger changes to the algorithm.

Earlier papers used OpenGL. This seems to be not an efficient option anymore, since CUDA was release in 2007.

Power consumption higher compared to FPGA

CPU

Different approaches for optimizations. In general reduction of complexity of the algorithm without specific adaptations to hardware components

Instantiation of the recommended implementation method in the reference platform

Instantiation of the recommended implementation method (based on GPGPU) can be done in the next generations of TULIPP reference platform only, because in first instantiations no GPU will be available.

Evaluation of the guideline in reference applications

Evaluation of the guideline in the reference application is planned while the TULIPP project.

9.2 Choosing a Real-time Operating System

Guideline advice

The RTOS should be chosen based on the application requirement. You need to choose the RTOS to be used based on the requirements regarding performance, safety/reliability, timing, resources utilization and required devices and services. If your design requires high performance, meaning that you need to use a multicore CPU and/or hardware accelerators (e.g. FPGAs), it is recommended to use an RTOS that supports those devices.

If your design needs a high degree of reliability, you have to choose an RTOS that can provide that level of reliability with features such as time & space isolation, watchdogs or self-repair. If you need to satisfy some industrial safety norm, you need to make sure that the RTOS is certifiable at the required level. Otherwise, a more common, non-safety certifiable RTOS will be enough.

If your design requires hard real-time behaviour, then you need to choose an RTOS that can warranty on time execution by design and never rely on having a performance that is “good enough on average”. Average timings are anecdotic for a real-time system: it is the worst case execution time that must be met. It is also recommended to estimate the effect of latencies and jitter on system performance.

The choice of RTOS must be made according to the hardware constraints of the target platform regarding memory, power and other limited resources. In particular, the footprint of the RTOS and middleware can have a significant impact if memory is limited, or even when not limited the fact that the RTOS kernel is small means it could be cache resident so that OS

overheads is minimal. This can be a major different when compared to large general purpose Oss. Regarding power, support for low power features (e.g. DVFS) at the RTOS level is recommended for low power designs.

The chosen RTOS must be able to run on the target platform and drivers for the required devices and services should be either available or foreseen as part of the development. It is recommended to limit device and service support to the strict necessary and not to clutter the system with support for unnecessary devices. This is the reason to recommend an RTOS with a modular micro kernel architecture.

Audience of the guideline

Application developers; System architects; Operating system designers

Author and guideline responsible

Antonio Paolillo (HIPPEROS)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Operating System.

Insights that led to the guideline

This recommendation is based on theoretical and experimental background information regarding real-time systems for critical applications.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

This guideline has to be followed before implementation. It is recommended to make a short list of possible RTOS choices and compare different products based on the criteria given in the recommendation and on their impact on the final design.

Instantiation of the recommended implementation method in the reference platform

The choice of RTOS for the reference platform (HIPPEROS) was made based on this recommendation taking into account the requirements for the TULIPP use cases.

Evaluation of the guideline in reference applications

Evaluation of the guideline in the reference application is planned while the TULIPP project.

9.3 Do Not Turn the FPGA Device On and Off too Frequently

Guideline advice

Turning FPGA devices off to save active power may turn out to be counter-productive. FPGA devices can draw a high amount of power during start-up. This can deplete energy in batteries quickly if the device is powered on and off repeatedly.

Audience of the guideline

Application developers

Author and guideline responsible

Ananya Muddukrishna (NTNU)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Low-power FPGA design, power measurement

Insights that led to the guideline

It is traditional wisdom to turn off unused GPUs and CPUs to save power. However, this wisdom does not transfer to FPGA designs since they have to be reconfigured when powered on. Reconfiguration consumes extra power.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Measure the currents and power drawn when the FPGA device is powered on, being configured, in standby, and active. Include these measurements in system design and dynamic power management decisions.

Power-on current can be very high if the power-on sequencing is incorrect or the temperature is outside the recommended range. Configuration current can be higher than active power for low-power designs. SRAM-based FPGA devices draw current to restore configuration from non-volatile memory. Standby current is drawn all the time the device is powered on and active current when the device is active i.e., performing computation.

Instantiation of the recommended implementation method in the reference platform

Not all vendors provide power measurement capability on-board. This is true for the EMC2DP, a predecessor for the reference platform. NTNU is constructing a general-purpose, fine-grained, high-frequency power monitoring device for the EMC2DP. This device will enable programmers to easily measure currents and power drawn when the FPGA device is powered on, being configured, in standby, and active. We expect to adopt the device to the reference platform (when available) in case it does not contain an equivalent power monitoring capability on-board.

Evaluation of the guideline in reference applications

Evaluation is planned within RTOS power management, within FPGA device selection, and within reference applications. When done, we will report evaluation in detail.

9.4 Move Data Processing as Close to the Sensor as Possible

Guideline advice

Move data processing as close to the sensor as possible. (Pre-) Processing a video stream with an embedded CPU consumes too much processing power, time and energy. KPIs in your real-time enabled low-power image processing platform will not be met.

Audience of the guideline

Application developers

Author and guideline responsible

Magnus Peterson (Synective Labs)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Image processing, low-power, sensor data, video stream

Insights that led to the guideline

In the TULIPP reference platform, sensor data can be connected to the processing system or to the FPGA part directly. (Pre-) Processing the sensor data in the FPGA, reduces latency and increases throughput, due to the parallel nature of FPGAs.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

If available, use on-board methods provided by the hardware manufacturer. For example, for the TULIPP starter kit, connect the sensor to the FPGA part directly. Consider the following examples of a Full HD video processing system with a simple pass-through and an HDMI output.

Instantiation of the recommended implementation method in the reference platform

1. A USB camera connected directly to the ARM processor of the Zynq SoC as input source.
2. Using an HDMI input connected to the FPGA of the Zynq SoC the pass-through design.
3. With optimizations of the part of the data processing responsible for handling the sensor input the frame rate (e.g. using VDMA instead of DMA).

Evaluation of the guideline in reference applications

1. This achieved 2 frames per second or even less. The Linux operating system uses OpenCV to read the input and writes it directly to the HDMI output.
2. This achieved 32 frames per second. The Linux operating system only monitors the processing of the video stream. Here, the sensor is moved close to the data processing executed on the FPGA only.
3. This achieved up to 60 frames per second.

This shows that moving the sensor as close as possible to the data processing – combined with simple optimizations – can improve your system.

9.5 Should I Use a FPGA or an ASIC?

Guideline advice

The choice of an FPGA for an Image Processing system has benefits in terms of flexible Input/Output, as can implement the entire camera standard found in Table 1 of D1.1.

An FPGA can also provide pre/-post processing on images. It's actually possible to add 32-bit and/or 64-bit CPUs inside a bare-bone FPGA and that comes typical as an IP-Core, but it does use a lot of the FPGA resources.

This is a route to take if your aim to develop a system for eventual volume production, as then possible to create an "Application-Specific-Integrated-Circuit" [ASIC]

Audience of the guideline

Application developers; Firmware developer

Author and guideline responsible

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Critical components selection, i.e. the central processor unit (CPU)

Insights that led to the guideline

If you have chosen an FPGA (Zynq SoC) like the TULIPP Starter Kit, you should check what IP cores are available and take them into account in your toolchain and development environment.

In the examples and Use-Cases for TULIPP, we have a CPU integrated into the FPGA by default, so the IP Cores we refer to here will be for the FPGA fabric.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

The biggest problem will be to find a suitable IP-Core, so below are suggestions only:

1. In 'software-land', then it's possible to find lots of algorithms and subroutines for even the most demanding image processing applications and many are included in textbooks and free. Publisher, like Wiley, has more than 40 (forty!) titles on the subject of image processing - http://eu.wiley.com/WileyCDA/Section/id-420274.html and the OpenCV community - http://opencv.org/ - has thousands of programs that are BSD licensed and free.
2. What about 'hardware-land'? Well, not many, as highly specialized. The best place is https://opencores.org/. Have 270000+ registered users. The free IP-Cores are typical hardware specific interfaces, rather than Image Processing specific, so restricted and limited use for the TULIPP STARTER KIT.
3. The handful of FPGA vendors that sells devices to the open market have their owe IP-Cores and you can find them on their respective websites, but they are always locked to the specific vendor and sometimes even to specific devices. Seldom comes with sources, hence can't be enhanced or reduced. They are typical also free, like this example from Xilinx - https://www.xilinx.com/products/intellectual-property/ef-di-vid-img-ip-pack.html, so difficult to complain. Never seems to support the LATEST vendor specific tool-chain, hence not very useful.
4. Independent vendor IP-Cores, like BarcoSilex's - https://www.altera.com/solutions/partners/partner-profile/barco-silex.html - are more expensive, but comes with support and typical also portable to next generation of FPGA. Options are for 'closed' IP that is sold per use/chip/system, as modern FPGA has unique ID option or full sources and unlimited usage. These IP vendors typical support cross semiconductor cores and also offers Verilog options to enable ASIC implementation
5. One important step in the development of algorithms for FPGAs is the move by vendors towards "C-to-VHDL" in different ways and routes. It's possible to get a software algorithm in either OpenCL, OpenGL, C or C++ (and more options coming) and convert it to target FPGA fabric. This is what TULIPP uses for the starter-kit and enables porting of current applications that can be tested on a CPU and moved to FPGA
6. The high-level Model programming of FPGA is also possible, but expensive. Tools like MathWork's Simulink - https://uk.mathworks.com/solutions/fpga-design.html [vendor agnostic] and NI's LabVIEW FPGA http://www.ni.com/labview/fpga/ [vendor specific] can create executable code that can be ported to the FPGA found on TULIPP STARTER KIT.

Instantiation of the recommended implementation method in the reference platform

In terms of priority of approach, then it depends on budget, timescales and personal preference, but selecting an independent vendor to support your development is the most efficient and highly recommended. Plenty of vendors are available.

Evaluation of the guideline in reference applications

The TULIPP toolchain will contain a few examples of IP-Cores that have been selected to support our Use-Cases.

9.6 Reordering Loops can Reduce Bandwidth Requirements

Guideline advice

Many image processing algorithms consist of several layers of nested loops.

By rearranging the order of the loops, you can in many cases reduce the required memory bandwidth and thus increase the processing speed. Try to find the loop-order that let the inner loop(s) work with local data.

For CPU implementations, this means to maximize the utilization of the memory cache, for FPGA based systems it means to find a structure where most data accesses use the FPGA internal memory.

Audience of the guideline

Application developers

Author and guideline responsible

Philippe Millet (Thales)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Code optimization

Insights that led to the guideline

There is a large difference in memory bandwidth and latency for internal FPGA/cache memory compared to external accesses, so the more the cache/internal FPGA memory can be used, the better.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Carefully evaluate the memory access patterns of your algorithm.

If an FPGA is your target, try to understand how much of the internal memory that can be allocated to this part of the algorithm. Try to rearrange your algorithm so that the data you store externally only needs to be streamed through once, and the data you access more frequently is stored in FPGA internal memory.

Specifically, also try to keep data that has a random access pattern to internal memories (or caches for CPUs) while data that is sequentially read can be stored externally, and only scanned through once.

Instantiation of the recommended implementation method in the reference platform

This guideline will be addressed in all use case implementations.

Evaluation of the guideline in reference applications

Memory utilization and algorithm processing speed will be monitored and evaluated during the implementation phases.

9.7 Upgrading to newer parts/FPGA architectures

Guideline advice

Upgrading the system, replacing the FPGA for another one that belongs to another family, or a newer part, may reflect minor changes that can damage the hardware in case the user doesn't take the appropriate care.

Audience of the guideline

Hardware Designers; System architects

Author and guideline responsible

Timoteo Garcia Bertoa (Sundance)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

hardware, upgrade, FPGA architecture

Insights that led to the guideline

Hardware upgrades can cause incompatibilities in software due to configuration files, which are recognizable and therefore easy to fix, but sometimes can lead into damaging the hardware. From 7 series to Ultrascale, the distribution of the logic varies, making some types of logic interface using different IO standards, requiring different range of voltage. Likewise, even from one FPGA part to another, sharing the same pinout, FPGA banks might be different, and accept different ranges of voltage. This can make the user think its applying the correct voltage when in reality is killing the device.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

The user should always take in consideration the different approaches a new family of FPGAs or a new part have when interfacing I/O. Software can detect incompatibilities when trying to write a bitstream for a different part, assuming the same pinout, recognizing different I/O standards or incompatible FPGA banks, but not always. The physical configuration in hardware through jumpers, etc, is as important as the appropriate changes in software to adapt the new hardware to the system.

Instantiation of the recommended implementation method in the reference platform

Moving from Z7015 to Z7030, the conditioning of the FPGA on Trenz modules makes both of them seem exactly the same, promising better performance and much more resources in Z7030. In reality, despite the pinout is the same, TE0715-30 has High Performance and High Range banks, while TE0715-15 only High Range. Applications where 3.3V are applied to Z7015, can be easily ported to Z7030, but always changing the I/O standards, to support up to 2.5V in the HP banks of the FPGA. Applying 3.3V on the Z7030 would damage its HP banks. Moving from TE0715-30 to TE0820 (Ultrascale+). would mean HR+HP banks to only HP banks. 3.3V should never be applied to TE0820. In the same way, the corresponding board files that the user imports in Vivado, can have default configurations, applying incorrect voltages if the

board files have been replicated. Board files of EMC2-7015 and EMC2-7030 should be not only different at the Zynq configuration, but avoiding 3.3V in all the I/O interfaces for the Z7030.

Evaluation of the guideline in reference applications

9.8 Optimize nested loops with early exit by measuring the number of unconditional iterations

Guideline advice

Loops with early exit that are themselves within other loops should be profiled to see if the early exit always happens after a certain number of iterations. If so, the ordering of the loops can be changed to allow for streaming those initial iterations.

Audience of the guideline

Application developers;

Author and guideline responsible

Carl Ehernstrahle (Synective Labs)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Code optimization

Insights that led to the guideline

The implementation of Viola-Jones in the Automotive use case is sped up by two orders of magnitude by using a Cascade. This means that after each stage in the detection, if the score is below a threshold, the detection is false and the loop exits. There are 2000 stages in total but 95% of detections exit before stage 100. However, this is not easy to implement on an FPGA. Thinking about this problem led to this insight.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

1. Figure out the number of iterations that are always done. In the automotive use case, no detection exited before 20 stages.
2. Split the loops into two parts: early and late stages.
3. Change the loop ordering in the early stages.

Example:

```
for x from 0 to 640
    for stage from 0 to 2000
        do classifier stage
        if score below threshold then break
```

becomes

```
for stage from 0 to 20
    for x from 0 to 640
        do classifier stage

for x from 0 to 640
    for stage from 21 to 2000
        do classifier stage
        if score below threshold then break
```

The first part can be streamed in this case, and the second part can avoid much of the work.

Instantiation of the recommended implementation method in the reference platform

This will be implemented in the automotive use case when hardware acceleration work is started.

Evaluation of the guideline in reference applications

It will be evaluated when it has been done.

9.9 Source Code Organization

Guideline advice

In projects involving both SW and HW, it is wise to consider how the code is organized into different source files. Done correctly, the build process will be much faster.

Audience of the guideline

Application developers;

Author and guideline responsible

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Toolchain

Insights that led to the guideline

It is common practice to organize source files according to functionality, and the HLS capabilities of SDSoC suggests that this is acceptable. However, rebuild times will really suffer if done without care.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Normal source code conventions still applies, but an additional convention should be adopted: Put HLS code into source files of their own. This means both HW accelerated functions and code that directly calls these functions.

The motivation is that the HLS tool will require a rebuild of the FPGA bitfile every time source files with HLS code are touched. This is very time consuming. By organizing the source files correctly, changes to the SW portion of the application will only require a recompile of the ELF file, which is usually very fast compared to rebuilding the bitfile.

Instantiation of the recommended implementation method in the reference platform

Example applications follow this guideline.

Evaluation of the guideline in reference applications

9.10 Automating the Toolchain

Guideline advice

Automate as much as practically possible with scripts.

Audience of the guideline

Toolchain designers;

Author and guideline responsible

Ananya Muddukrishna (NTNU)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Toolchain

Insights that led to the guideline

Recommended implementation method of the guideline along with a solid motivation for the recommendation

While GUI IDEs are convenient for certain operations and for certain stages of the tool learning process, sometimes productivity can be enhanced by switching partially or fully to command line tools and scripting languages. Most development tools, such as Xilinx SDSoC, can be completely controlled from the command line and scripted with TCL. Time should be spent to learn these capabilities.

Examples of operations suited for scripts:

1. Repetitive and/or error prone operations. Example: Regression testing a new build on the HW platform, when interactive debugging capabilities are not needed.
2. Complex operations where the steps are unlikely to be remembered to the next time it is required. Example: Running an experiment on the board and creating reports from program runs
3. Using the toolchain on systems where a GUI is impossible or impractical. Example: The toolchain is installed on a build server and the network connection to the developer desktop is too slow for comfortably forwarding the GUI.

Instantiation of the recommended implementation method in the reference platform

Makefiles, deployment and debugging scripts are supplied for all example programs.

Evaluation of the guideline in reference applications

9.11 Timestamp ADC samples promptly

Guideline advice

When sampling an analog signal using an ADC, timestamp the samples as close as possible to the sampling instant.

Audience of the guideline

Hardware Designers; System architects

Author and guideline responsible

Ananya Muddukrishna (NTNU)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Sampling, performance analysis

Insights that led to the guideline

A free-running ADC enables sampling an analog signal at a high frequency. Samples from the ADC are typically sent over a serial link to a host computer for timestamping and further processing. These timestamps do not capture the timing nature of the analog signal accurately due to cable latency, buffering by the serial communication stack, and unpredictable OS scheduling of the processing program.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Timestamp ADC samples promptly i.e., as close as possible to the ADC sampling instant and before sending the over the serial link to the processing program on the host computer. This ensures that the timestamps reflect the timing nature of the analog signal.

Instantiation of the recommended implementation method in the reference platform

The current sensor board called Lynsyn built to measure power consumption of the EMC2DP timestamps ADC samples before sending over serial link to the host computer.

Evaluation of the guideline in reference applications

Compare timestamping accuracy for TULIPP reference application X for two cases: timestamping at host-side and timestamping at current sensor-side.

9.12 Save implementation time by delegating customizations to the hardware vendor

Guideline advice

Save implementation time by delegating customizations to the hardware vendor

Audience of the guideline

Hardware Designers; System architects

Author and guideline responsible

Ananya Muddukrishna (NTNU)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Component selection, embedded hardware design

Insights that led to the guideline

Designers can decide to implement missing hardware features on their own. However, custom hardware implementation is a surprisingly difficult and time consuming process since it requires careful component selection, future-proof interfacing, and extensive testing.

The root problem here is overlooking the possibility to use the hardware vendor to make customizations. Vendors are often perceived as too expensive to approach for customizations. Experience says the opposite is true. Vendors do make custom changes, often at an affordable price point if they can reuse existing designs. At least, vendors can confirm if the required customization is a good idea at all and suggest alternatives.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

If a piece of hardware does not have required features, talk to your vendor before starting to implement the missing features on your own. If the vendor's terms for customization are reasonable, delegate implementation of the missing features to the vendor.

Instantiation of the recommended implementation method in the reference platform

We delegated modification X to the hardware component Y in the TULIPP reference platform Z to vendor V.

Set 1:

- X = Precision shunt resistors on PS and PL power lines
- Y = UltraScale+ MPSoC SoM
- Z = TULIPP-PI1
- V = Trenz Electronics

Set 2:

- X = 1Msps ADC modules for sampling current drawn by PS and PL
- Y = UltraScale+ MPSoC SoM
- Z = TULIPP-PI1
- V = Trenz Electronics

Evaluation of the guideline in reference applications

It is unclear how to evaluate this guideline. Perhaps a case-study of DIY Vs. vendor customization.

9.13 Selecting PSU to supply the chosen hardware

Guideline advice

The user should choose the appropriate PSU when testing the corresponding application on the hardware. For low-powered boards, PC PSUs are not always providing the correct voltages.

Audience of the guideline

Hardware Designers;

Author and guideline responsible

Timoteo Garcia Bertoa (Sundance)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Power, PSU, hardware

Insights that led to the guideline

Some of the partners within the TULIPP project have faced the issue of finding "hardware not working", when trying their use-case on the TULIPP board. The solution provided in many cases was replacing the PSU for another one that provides the correct voltages, or add a complementary load to pull from the PSU enough current. (i.e. connecting a hard disk to one of the sata connectors of the PSU)

Recommended implementation method of the guideline along with a solid motivation for the recommendation

The user should always have in mind that the TULIPP board expects 12V,5V and 3.3V as input. From the PSU to the power connector onboard, sometimes there is as a voltage loss. As well, the PSU may be providing not enough voltage to the board, requiring more load, or having to be replaced by another PSU. To find this out, the user should measure the voltage onboard, using a multimeter or any other instrument capable of this task.

Instantiation of the recommended implementation method in the reference platform

The EMC2 has JP8,JP7 as reference for the voltages. Depending on the position, the user can measure 3.3V, 1.8V and 2.5V. 5V can be measured at the power connector. In case of using a PCIe/FMC application, 12V can be measured at the power connector too. If the voltage measured on the 3.3V pin results less than 3.2V, the user should suspect a possible miss functionality of the board, and try to achieve a proper level.

Evaluation of the guideline in reference applications

To be done during hardware integration and use cases implementation

9.14 Manage to accelerate streaming capable functions in a single accelerators

Guideline advice

Fuse as much streaming capable functions into a single accelerator as possible, to reduce the resource and latency overhead.

Audience of the guideline

Application developers;

Author and guideline responsible

Carl Ehernstrahle (Synective Labs)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Code optimization

Insights that led to the guideline

Streaming between functions reduces the memory bandwidth consumption and decreases the computation time for applications implemented of FPGAs. One methodology of how to implement this is by implementing an accelerator for each function of an application and letting the tool handling the dependencies. The problem is that this adds unnecessary resources to the design, which are needed to connect each function to the surrounding system. This problem can be solved by connecting the functions within a single accelerator using loop/function level parallelism and a FIFO.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

The different functions are designed to be streaming capable using the DATFLOW pragma and can be connected to each other with a FIFO. The following code example shows how to connect the streaming capable functions func1 and func2.

```
void func0(int *input, int *output) {

static int buffer[PIXELS];

#pragma HLS STREAM variable = buffer depth = 512

#pragma HLS DATAFLOW

func1 (input, buffer);

func2(buffer, output);

}

}
```

Instantiation of the recommended implementation method in the reference platform

Implementation examples are provided by the image processing library.

Evaluation of the guideline in reference applications

This guideline will be evaluated during use cases implementation.

9.15 Integrating your image processing HDL code with the platform

Guideline advice

If you have a handcrafted HDL code that can do part of the processing, you want to integrate it with your SDSoC design in order to enhance acceleration and resource usage (well Handcrafted HDL is more resource efficient than High-Level Synthesis).

Audience of the guideline

Application developers; Firmware developer

Author and guideline responsible

Lester Kalms (RUB)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Hardware design, design optimization

Insights that led to the guideline

Feature-based Simultaneous Localization and Mapping (SLAM) algorithm have as the first processing step the feature detection and description task. They're computationally intensive and can operate at a streaming of Data. Thus they're suitable for FPGAs. The availability of HDL based implementations for some functions like Harris or FAST corner detection algorithms require different methods if we want also to accelerate other part of the algorithm using SDSoC.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

The implementation of such a design depends on the requirement of the application. Certain applications need as an input the processed data and the image data. In that case creating the Vivado platform require both AXI Video DMA(VDMA) for image acquisition and DMA for processed data (in the case of ORB descriptor, it would be the corner position and the binary descriptor of the corner).

Instantiation of the recommended implementation method in the reference platform

There are two kind of Cameras used in TULIPP's projects, HDMI-based and Cameralink-based Camera. The combination of EMC2 and Avnet HDMI Input/Output FMC was used in a Sobel and motion detection demo. Integrating HDL with this platform require finding-out the stream of pixels and the synchronization signals. One way to do that is using the signals out of the Avnet VHDL module fmc_imageon_hdmi_in.vhd (de, vblank, hblank and data). Note that 'vblank' and 'hblank' are different from 'vsync' and 'hsync'.

Evaluation of the guideline in reference applications

Not yet evaluated.

9.16 Use hil testing rather than wait for a final hardware

Guideline advice

Use Hardware in the loop (HIL) testing with early prototypes rather than wait for fully integrated hardware.

Audience of the guideline

Application developers; System architects

Author and guideline responsible

Igor Tchouchenkov (Fraunhofer)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Testing, drones, project/time management

Insights that led to the guideline

The emc2 board uses bulky power supply and is lacking of hardware interfaces like an LVTTL serial port. In order to make it *flyable* several steps are necessary. But in order to test a collision avoidance algorithm, you need to have a flying drone as you need to have closed loop tests.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

If you use a drone simulation to test your algorithms you can do this in parallel with the integration work and save time. This also gives you the advantage of a higher testing density as you are not limited to the Weather or risks of drone crashes.

Instantiation of the recommended implementation method in the reference platform

drone use-case HIL version

Evaluation of the guideline in reference applications

9.17 How to get EMC2 Board communicate with an UAV

How to get EMC2 Board communicate with an UAV

Guideline advice

The most important step for getting the EMC2 Board to work with an UAV is to establish a connection. There are different ways to do this.

Audience of the guideline

System architects

Author and guideline responsible

Fraunhofer

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

UAV control, serial I/O, UART

Insights that led to the guideline

The TULIPP platform provides an environment for low power and high performance image processing. Especially users in the field of robotics can benefit from environments like this. Therefore it is really important to offer help for easy implementation of I/O ports.

Access the UAV through the Serial Port

Setting up in Vivado

First of all you need to build a new Hardware Platform where you define all your in/- and outputs. The Zynq 7030 uses different MIO Ports which can be made „external“ by changing to EMIO in Vivado.

- By double clicking the "Zynq Processing System Block" you can make for example the UART1 Port EMIO. Right click on the UART1 Port in the Block Diagram and you can select "make external".
- In this step you decide which output you prefer. You can use the UART as an RS232 output or you can choose an 1.8V TTL output. Unfortunately 3.3V is not possible because of the dependency to a High Performance Bank on the FPGA. These banks only support up to 1.8V on the Zynq 7030.
- Next you define the constraints by hand or you use the I/O Planning section after the synthesis to define the ports of the newly generated outputs with the help of Vivado. Run the implementation and generate the bitstream. To use the new hardware design you need to export the .hdf file including the bitstream.

Building the distribution with Petalinux

Petalinux is a tool which is based on „Yocto“ where you can rapidly generate your own distribution for testing your hardware. Please be aware that this distribution is not fully recommended by TULIPP but since HIPPEROS is still in development it can be a good alternative

for testing. Information about how to setup a Petalinux Environment and how to use Petalinux can be found inside the Xilinx Documentations.

- At first create a new project and get the hardware description file from Vivado to synchronize with the EMC2 Board. When this had been done the project can be configured.
- Include the libraries you want to use from the Petalinux „Filesystem Package“. - After doing that you can build your project. Petalinux creates different files which you need to combine into a BOOT.bin file and an image.ub file. These files need to be included into the BOOT folder of the sd card.

Generation of an executable file

To compile your Application you need to use Xilinx SDSoc or SDK.

- Create a new Application project and choose linux as the platform you want
- You need to select the stage folder in the Petalinux build which contains the rootfs of the new system
- You are able to compile your main, if your program depends on libraries or files you need to define them inside the build configurations.

Executing an Application

Place the 3 files in the Boot Folder of the sd card and power on the EMC2. While flashing the FPGA the green LED should light half a second and then turn of. If this is not the case probably there is something wrong with the generated bootfiles in Petalinux or Vivado.

- Connect the EMC2 Board with the UART0 to USB connector and establish a serial connection. In linux the board can be found on the /dev/ttyUSB0 port and uses a baudrate of 115200.
- Mount the sd card and execute the application.

Evaluation of the guideline in reference applications

This guideline will be evaluated in the UAV use case.

9.18 Use an external constant voltage reference for the ADC instead of the internal reference based on positive supply voltage (VDD)

Guideline advice

Use an external constant voltage reference device for the ADC instead of the internal reference based on positive supply voltage (VDD)

Audience of the guideline

Hardware Designers;

Author and guideline responsible

Ananya Muddukrishna (NTNU)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

ADC, measurement

Insights that led to the guideline

Positive supply voltage drifts with temperature and varies with load.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Use constant voltage reference chips such as LM385 or LT1009.

Instantiation of the recommended implementation method in the reference platform

Lynsyn, the power measurement board, part of the power measurement utility (STHEM), uses LT1009 as a constant 2.5V reference.

Evaluation of the guideline in reference applications

This guideline is evaluated during the development of the tool chain.

9.19 Isolate designs to low-power domains to reduce power consumption

Guideline advice

Isolate functionality to power domains to reduce power consumption

Audience of the guideline

Application developers; Firmware developer

Author and guideline responsible

Carl Ehernstrahle (Synective Labs)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Power measurement, power optimization

Insights that led to the guideline

Zynq UltraScale+ MPSoC has three power domains -- full-power, low-power, and PL domains. Each domain has components that can be clock-gated. Not using entire power domains saves more power than using all domains with optimized clock-gating.

Reference: https://www.xilinx.com/support/documentation/white_papers/wp482-zu-pwr-perf.pdf

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Constrain applications to work with low-power domains.

Instantiation of the recommended implementation method in the reference platform

TULIPP project applications X, Y and Z avoid the full-power domain entirely, and use low-power and PL domains.

Evaluation of the guideline in reference applications

This guideline is evaluated during the implementation of the use cases.

9.20 Beware the parameters of external data/clocking when testing or debugging

Guideline advice

Beware the parameters of external data/clocking when testing or debugging, as it can lead to misunderstanding when the problem resides on the external patterns provided to the system in order to debug.

Audience of the guideline

Hardware Designers; System architects

Author and guideline responsible

Timoteo Garcia Bertoa (Sundance)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Debugging, testing

Insights that led to the guideline

The EMC2 provides an external SMA input connector (J5), to apply external clocking/data. One of the ways of scoping internal signals in Vivado is instantiating an ILA core (Integrated Logic Analyzer), which needs a clock input that belongs the clock domain of the signals the user wants to scope. Certain designs run under clock domains that are not appropriate for the user to use the ILA core, as the frequency of the sampling clock for the core might be too high, and therefore, to scope a certain range of the signals the depth of the ILA must be very high, which

is a limitation on the size of the FPGA in terms of resources. An easy way to scope internal signals of a design that run under a very fast clock, assuming the user just needs to see the behaviour of these, is using an external clock, asynchronous to the signals, but easy modifiable, being the user able to inject the desired frequency to increase/reduce samples shown in the ILA for the same amount of resources in the FPGA.

When injecting an external clock through J5 in the EMC2, the user must remember that this connector is connected straight away to an IO pin of the FPGA, and therefore, if the external clock is generated from a signal generator with internal 50Ohms, the amplitude of the clock might not be enough for the FPGA to detect it as a clock input. This will make the ILA core never run, and therefore the user may blame the design without noticing the ILA core is not fed properly with a free-running clock.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Always measure the voltage levels, frequency and amplitude of the external clocks applied to an FPGA in order to test, to discard the possibility of reduced amplitude in an IO pin, avoiding the user being led to a wrong diagnose.

Instantiation of the recommended implementation method in the reference platform

When using J5 to use external clocks on the EMC2, make sure the amplitude is 1.8, 2.5 or 3.3V peak to peak (depending on the IO standard applied through the jumpers) at the output of your signal generator, regarding the possible internal resistors within the device, so that the FPGA receives the correct levels into the corresponding IO pin.

Evaluation of the guideline in reference applications

9.21 Make sure to access streamed data from within accelerated function

Guideline advice

When using streamed data as in/output to a hardware accelerated function, it is important to perform a read/write operation in each loop operation. No conditional access to the streamed data.

Audience of the guideline

Application developers;

Author and guideline responsible

Thales

Category of the guideline

code optimization, FPGA design, HLS

Insights that led to the guideline

The stereo algorithm for the UAV use case requires a simultaneous input of two images. A first test, was to load two images and display the upper half of the left image and the lower half of the right image simultaneously. The data of the input images was streamed into the accelerated function and accessed conditionally depending on the current row index. Executing the code with synthesized HW functions only showed a black screen, as the board didn't start up properly. Remedy was to access all input data within each loop iteration and perform a conditional variable selection.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Example:

```
func(unsigned char* in_left, unsigned char* in_right, unsigned char* out)
{
 int x, y;
 for(y = 0; y < IMG_HEIGHT; y++)
 {
  for(x = 0; x < IMG_WIDTH; x++)
  {
    unsigned char px;

    // wrong access
    if(y < (IMG_HEIGHT / 2) )
      px = in_left[y * IMG_WIDTH + x];
    else
      px = in_right[y * IMG_WIDTH + x];


   out[y * IMG_WIDTH + x] = px;
  }
 }
}
```

Becomes:

```
func(unsigned char* in_left, unsigned char* in_right, unsigned char* out)
{
 int x, y;
 for(y = 0; y < IMG_HEIGHT; y++)
 {
  for(x = 0; x < IMG_WIDTH; x++)
  {
    unsigned char px_l, px_r, px_out;

    // correct access
    px_l = in_left[y * IMG_WIDTH + x];
    px_r = in_right[y * IMG_WIDTH + x];

    if(y < (IMG_HEIGHT / 2) )
      px_out = px_l;
    else
      px_out = px_r;

   out[y * IMG_WIDTH + x] = px_out;
  }
 }
}
```

Instantiation of the recommended implementation method in the reference platform

Instantiation as part of the UAV use case.

Evaluation of the guideline in reference applications

Evaluation as sandbox test application

9.22 Writing for HW

Guideline advice

To keep the possibilities for HW/SW partitioning as open as possible, compute intensive functions should be written with HLS in mind. Optimizations for execution on a CPU or an FPGA should come towards the end of the application development process, after the HW/SW partition has been fixed.

Audience of the guideline

Application developers; Firmware developer

Author and guideline responsible

Ananya Muddukrishna (NTNU)

Category of the guideline

Toolchain

Insights that led to the guideline

Work on an automatic design space exploration tool has shown that it is non-trivial to convert a typical C function to a form suitable for HLS. Manual work is typically needed, sometimes resulting in large rewrites.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

High level synthesis makes it possible to write HW modules to be executed on an FPGA using normal C code. This is useful for SW programmers without HDL experience which then can produce FPGA modules without learning a new language. It is also useful for experienced FPGA developers when full control over the resulting module is not needed or wanted.

The programmer cannot, however, write code without considering the limitations of the HLS tool. In the early phases of application development, the final HW/SW partitioning of the application is likely still unknown. By keeping HLS in mind, less work will be required when C functions are later chosen for HW implementation.

The following guideline increases the possibility that your code will be HLS compatible:

1. Organize your code such that the compute intensive parts are self-contained kernels. Keep system and library calls elsewhere.
2. HLS requires that C constructs are of a fixed or bounded size. Either use only fixed or bounded sizes in the compute kernels, or write the code such that it is easy to later fulfil this requirement.
3. Inside the compute kernels, memory allocation should be avoided and stack allocated variables should be preferred. When more memory is needed than can be safely

allocated on the stack, #ifdef can be used to select between implementations using malloc and stack variables.

4. Avoid pointer casting, and if needed use only pointer casting between native C types
5. When using pointer arrays, make sure the pointers point to a scalar or array of scalars. Avoid pointer arrays pointing to additional pointers.
6. Do not make the compute kernels recursive

For further information on writing HLS compatible code, see Xilinx User Guide UG902.

Instantiation of the recommended implementation method in the reference platform

Evaluation of the guideline in reference applications

9.23 Remove recursion when aiming to parallelize code

Guideline advice

Use loops instead of recursive function calls when aiming to optimize code for parallel programming on appropriate hardware such as GPU or FPGA.

Audience of the guideline

Application developers; Firmware developer

Author and guideline responsible

Ruf, Boitumelo (Fraunhofer)

Category of the guideline

code optimization, GPGPU, FPGA, HLS

Insights that led to the guideline

A key-aspect of the SGM algorithm is the aggregation along concentric paths which centre in the currently processed pixel. In [1] a recursive path traversal is proposed. Yet, recursive function calls are very inefficient when it comes to running on parallel hardware. Moreover APIs such as CUDA, OpenCL or Vivado HLS do not support recursive function calls.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Hence, instead of implementing recursive function calls use loops instead. This requires temporary buffers or variables to store the parameters which are passed to the recursive function between consecutive loop iterations.

Strategies on how to convert recursive functions to loops are found here.

Instantiation of the recommended implementation method in the reference platform

Such a conversion is instantiated in optimizing the SGM algorithm for parallel hardware as part of the UAV use case. The recursive path traversal of the aggregation is replaced by iterative strategy.

Each path direction is implemented as a separate loop going from one image boarder to the other. For each pixel along the path the aggregated costs are stored in a separate cost volume. This allows a parallelization of the path traversals as all paths are independently operating on constrained subsets of data.

See UAV use case code for more details.

Evaluation of the guideline in reference applications

This guideline will be evaluated during use case implementation.

Related guidelines

How to optimize sgm for GPGPU

References

[1] Hrischmueller, H., "Stereo Processing by Semi-Global Matching and Mutual Information", Transactions on Pattern Analysis and Machine Intelligence (TPAMI), IEEE, vol. 30, pp. 328-341, 2008.

9.24 How to optimize sgm for GPGPU

Guideline advice

In order to efficiently map the path aggregation of the semi-global-matching (SGM) [1] optimization onto massively parallel hardware, such as GPUs, partition the aggregation into small kernels. Each kernel should operate on a locally constrained subsets of data. This enables the GPGPU driver to distribute the processing onto the large number of processing units inherent to modern GPUs, alleviating the SIMD paradigm of GPU programming.

Audience of the guideline

Application developers;

Author and guideline responsible

Ruf, Boitumelo (Fraunhofer)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

code optimization, SGM, GPGPU

Insights that led to the guideline

Implementing the algorithm for GPGPU requires a paradigm shift with respect to the path aggregation of the SGM optimization. A recursive path traversal starting from each pixel, for which the disparity is to be estimated, is very inefficient for massively parallel architectures. This is due to the fact that the image data is not accessed in a contiguous manner, making hardware optimizations very difficult. A redesign of the algorithm is required, removing the recursion in the path traversal.

Furthermore, each aggregation path only uses a locally constrained subset of data, i.e. the aggregated cost on the path itself. Hence, each aggregation path can run in parallel on different processing units

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Implement separate kernels for each path direction of the aggregation and instantiate each kernel with the number of paths which are to run for the given path direction.

Instantiation of the recommended implementation method in the reference platform

Instantiation as part of GPGPU implementation of the UAV-Use-Case.

Evaluation of the guideline in reference applications

This guideline will be evaluated during use cases implementation.

Related guidelines

Remove recursion when aiming to parallelize code

References

[1] Hrischmueller, H., "Stereo Processing by Semi-Global Matching and Mutual Information", Transactions on Pattern Analysis and Machine Intelligence (TPAMI), IEEE, vol. 30, pp. 328-341, 2008.

9.25 Save Intermediate Storage with Dataflow Regions

Guideline advice

When writing HW modules using Xilinx HLS, a streaming architecture is often the most efficient choice.

To avoid large intermediate buffers between sub-modules (which can easily require more than can fit in the FPGA), use the DATAFLOW pragma. Performance will most likely also improve.

Audience of the guideline

Application developers; Firmware developer

Author and guideline responsible

Timoteo Garcia Bertoa (Sundance)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

HLS

Insights that led to the guideline

Experimentation with Xilinx HLS examples.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Even though HLS gives the impression that any C code should work well, it is crucial for performance that the designer thinks HW architecture when designing.

One limitation of the FPGA compared to a CPU is available memory. On-chip memory is usually very limited. When designing a HW module where one sub-module is calculating a large intermediate result (e.g. a large image), and another sub-module continues on this intermediate result, a dataflow architecture (with the DATAFLOW pragma) will enable the sub-modules to run in parallel and reduce the necessary intermediate storage to only a one-element FIFO (e.g. holding a single pixel).

Instantiation of the recommended implementation method in the reference platform

Evaluation of the guideline in reference applications

This guideline will be evaluated during use cases implementation.

9.26 When to use conditional branching

Guideline advice

Conditional branching such as if-then-else is vital to most image processing applications, e.g. finding maximum similarity between pixels. While conditional branching is cheap on CPUs and FPGAs, it is to be used with caution when optimizing code for parallel GPGPU.

Audience of the guideline

Application developers; Firmware developer

Author and guideline responsible

Ruf, Boitumelo (Fraunhofer)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

code optimization, GPGPU, FPGA

Insights that led to the guideline

CPUs are designed for general purpose processing and are equipped with optimization strategies such as branch prediction which allow a fast response to conditional input. In order to achieve parallel processing on CPUs the programmer instantiates different threads and processes which can run concurrently on the different processing cores. The scheduler of the CPU is free to pause the processing of certain threads in order to react to important interrupts and inputs. Hence, it is not guaranteed that all threads will run synchronously. Furthermore, due to its flexibility the CPU, unlike GPGPUs, is able to only process the branch for which the conditional directive resolved to true.

FPGAs can also cope well with conditional branching, as HLS will create different paths for each conditional branch. The use of conditional branching is therefore very cheap when programming FPGA with the help of HLS.

In order to achieve great parallelism and high data throughput, GPGPUs run numerous (>100) kernels on a large number of processing units. The key aspect of this processing is that each

instantiation of the kernel is performing the same processing but on different subsets of data. GPGPU vendors call this paradigm Single-Instruction-Multiple-Threads (SIMT) which is similar to Single-Instruction-Multiple-Data (SIMD). This SIMT processing requires that all kernel instances run synchronously. This means that when kernels have conditional branching, all branches are evaluated and processed in order to keep the threads from diverging. At the end the result of the particular branch is chosen for which the conditional expression resulted in true. Hence, conditional branching with large bodies to save processing time is to be avoided, as all branches will be processed anyway. See [1] for more information.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

When optimizing code for GPGPUs try to use loops instead of if-then-else branches. Try to minimize conditional branching to inline conditionals such as: Variable_A = (Condition_for_A) ? Variable_B : Variable_C;

Instantiation of the recommended implementation method in the reference platform

Instantiation as part of GPGPU implementation of the UAV-Use-Case.

Evaluation of the guideline in reference applications

This guideline will be evaluated during use cases implementation.

References

[1] "CUDA C Programming Guide", Nvidia, 2017. [pdf]

9.27 Do not use floating point computation on FPGA

Guideline advice

Using floating point computation in FPGAs requires multiple loop iterations for basic arithmetic operations. Hence, in order to achieve high data throughput try to avoid floating point operations.

Audience of the guideline

Application developers; Firmware developer

Author and guideline responsible

Ruf, Boitumelo (Fraunhofer)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

code optimization, FPGA, HLS

Insights that led to the guideline

Using floating point penalties in the Semi-Global-Matching (SGM) algorithm led to low throughput, as HLS cores could not be pipelined, due to conversion from floating point to signed/unsigned.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Avoid floating point operations if possible.

Instantiation of the recommended implementation method in the reference platform

Instantiation as part of FPGA implementation of the UAV-Use-Case.

Evaluation of the guideline in reference applications

This guideline will be evaluated during use cases implementation.

9.28 Avoid LibTIFF library and use raw image format

Guideline advice

Avoid LibTIFF library and use raw image format

Audience of the guideline

Application developers;

Author and guideline responsible

Alvin Sashala Naik (Thales)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Image decompression libraries, hardware optimisation, cross-compilation

Insights that led to the guideline

Difficulties to integrate LibTIFF library to image compression and decompression for RTOS or bare metal implementation

Recommended implementation method of the guideline along with a solid motivation for the recommendation

During development phase, convert manually all test images in Tiff format into either PPM/PGM images or hexdump format to avoid dependency issues during cross-compilation for RTOS implementations.

Instantiation of the recommended implementation method in the reference platform

When testing the algorithm on a live image sensor, discard heavily opencv-dependent libraries and work on the raw data flow from the image sensor. Use C programming language so as to work according to Kahn Process Network programming rules.

Evaluation of the guideline in reference applications

This guideline will be evaluated during use cases implementation.

9.29 Use EMVA1288 to compare cameras

Guideline advice

Use metrics from the EMVA1288 standard to compare cameras

Audience of the guideline

System architects

Author and guideline responsible

Timoteo Garcia Bertoa (Sundance)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Device selection

Insights that led to the guideline

Selecting a camera that matches application requirements is a crucial first step. Mismatched cameras can complicate the processing pipeline and increase power consumption. Camera selection is a tiring and error-prone process since camera vendors use custom metrics that are difficult to compare even for experts. It literally a camera jungle out there!

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Use EMVA1288 metrics in datasheets to compare cameras. Avoid vendors that supply cameras without EMVA1288 or other standard comparison metrics.

Here is more information about EMVA1288.

Instantiation of the recommended implementation method in the reference platform

We selected camera A for TULIPP platform Z after comparing cameras A, B, C, D, E, and F using EMVA1288 metrics M, N, O, P, Q, and R.

Evaluation of the guideline in reference applications

Performance and power consumption degraded by X and Y respectively when we choose a bad camera B for application M. The best camera in terms of performance and power for M is A.

9.30 Low latency image processing over TCP IP data stream

Guideline advice

Real-time IP stream for sensor input and image processing

Audience of the guideline

System architects

Author and guideline responsible

Alvin Sashala Naik (Thales)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

TCP/IP, Gstreamer, OpenCV, FFMPEG

Insights that led to the guideline

Integration constraints from Thales Electron Devices (TED)

Recommended implementation method of the guideline along with a solid motivation for the recommendation

OpenCV structures aren't a good fit for real-time image processing over RTSP protocol (definitely not for real time critical applications). The best approach that would provide better configuration is running a GStreamer library pipeline through OpenVX/CUDA (with zero-copy) processing pipe and outputting the stream using GStreamer again. More importantly, one can use circular buffers which can be flushed every time and request the latest frame in the best approach. RTSP or MJPEG is not particularly helpful for hard real-time applications such as for medical imaging.

Instantiation of the recommended implementation method in the reference platform

Implement different pipelines using Gstreamer (one for HMDI display and another for storage on disk) over the RTSP stream. Perform image processing using OpenVX for zero-copy and low-latency.

Evaluation of the guideline in reference applications

This guideline will be evaluated during use cases implementation.

9.31 Major challengers to integrating hard real time image processing using neural networks in MPSoCs

Guideline advice

Integrate Convolutional Neural Networks in low-power embedded computer vision with hard real-time constraints

Audience of the guideline

Application developers;

Author and guideline responsible

Alvin Sashala Naik (Thales)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Neural Networks, Computer vision, Deep-learning

Insights that led to the guideline

Barriers to implement computationally intensive deep learning algorithms in low-power embedded system-on-chips considering hard real-time constraints

Recommended implementation method of the guideline along with a solid motivation for the recommendation

1. Reduce neural network weights to the order of kilobytes so that they fit into the SRAM memory (use lower than INT8 precision)
2. Network Pruning so as to make your neural network as most efficient as possible (up to 90% pruning possible and needed)

Instantiation of the recommended implementation method in the reference platform

The instantiation requires reworking the neural network which depends from the framework.

Evaluation of the guideline in reference applications

This guideline will be evaluated during use cases implementation.

9.32 Ensure Dataflow by using Inline Sub-functions

Guideline advice

Especially for smaller sub-functions, data streaming can become quite cumbersome. A simple alternative to ensure dataflow is to declare those functions as "inline".

Audience of the guideline

Application developers; software Developers;

Author and guideline responsible

Antonio Paolillo (HIPPEROS)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Code Optimization

Insights that led to the guideline

When trying to export code into sub-functions, the dataflow can be interrupted. Streaming the data in every function often comes with additional work. Inline-Sub-functions propose a simple alternative for reducing the complexity of the code, making it more comprehensible while also ensuring dataflow.

Recommended implementation method of the guideline along with a solid motivation for the recommendation

If streaming data to a sub-function appears to be too complex, try using: **#pragma HLS INLINE**

This removes the function as a separate entity in the hierarchy. After inlining, the function is dissolved into the calling function and no longer appears as a separate level of hierarchy in the RTL. In some cases, inlining a function allows operations within the function to be shared and optimized more effectively with surrounding operations.

However: An inlined function cannot be shared. This can increase area required for implementing the RTL. The function also does not show up in the hierarchy of the HLS report.

For more information about inlining subfunctions see SDx Pragma Reference Guide.

Instantiation of the recommended implementation method in the reference platform

Implementation Examples are provided in the referenced Guide above.

Evaluation of the guideline in reference applications

While Implementing the SGM-Algorithm, inlining subfunctions has been a solid method for exporting code into subroutines without hurting the pipeline. They can be used to help structuring the code.

9.33 Data logging using SD Card

Guideline advice

If process data needs to be stored (e.g. for post processing or verification) you can use the SD Card as a non-volatile storage.

Audience of the guideline

Application developers; System architects

Author and guideline responsible

Magnus Peterson (Synective Labs)

Guideline reviewers

Not yet assigned by the Quality Assurance Board.

Category of the guideline

Data Logging

Insights that led to the guideline

Taking on camera calibration forced us to store image information for post processing. Therefore, a method for storing was needed. Research led us to the following guide: https://embeddedcentric.com/data-logging-using-sd-cards/

Recommended implementation method of the guideline along with a solid motivation for the recommendation

Firstly, we need FatFs. This is a generic FAT/exFAT filesystem module for small embedded systems. More information can be found here: http://elm-chan.org/fsw/ff/00index_e.html

1. The corresponding file is included with: `#include "ff.h"`

2. Afterwards, an instance of type file system associated with the SD card needs to be declared: `FATFS fs;`

3. Also, an instance of a type file needs to be created: `FIL file1;`

4. In addition, we declare a variable which is used to hold the return value of the FatFs APIs:

```
FRESULT result;
```

5. Before we can open a file, we first need to initialize the SD Card by using *f_mount*:

```
result = f_mount(&fs, Path, 1);
if (result != 0) {
   xil_printf("Mounting failure: %d\r\n", result);
   return XST_FAILURE;
}
```

Path refers to a string pointer, containing the logical drive number: const TCHAR *Path = "1:"; We furthermore check the return value of *f_mount*, so we can derive possible errors. The corresponding error codes can be found on the FatFs site.

6. Now we can continue by opening the file with *f_open*:

```
result = f_open(&file1, "1:/L_img.txt", FA_CREATE_ALWAYS | FA_WRITE);
if (result != 0) {
   xil_printf("Opening failure %d\r\n", result);
   return XST_FAILURE;
}
```

"L_img.txt" specifies the name of the file, while the third parameter specifies the method of access. In this example, the file is created (first discared if existing) and WRITE-only.

7. Writing to the file is done with the *f_write* command:

```
result = f_write(&file1, (unsigned char*) ADDRESS, DATA_SIZE, &count);
if (result != 0) {
   xil_printf("Writing failure!\r\n");
   return XST_FAILURE;
}
```

ADDRESS is the pointer to the Array to be written. DATA_SIZE specifies the number of bytes to be written to the file, while count (unsigned integer) is used to count the already written bytes.

8. Finally, the file needs to be closed, which is done by using *f_close*:

```
result = f_close(&file1);
if (result != 0) {
   xil_printf("Closing failure\r\n");
   return XST_FAILURE;
}
```

Instantiation of the recommended implementation method in the reference platform

Evaluation of the guideline in reference applications

The guideline was used to save two images, each of the size 1024x768x2 Bytes, which equals 3 Mb. The storing process took around ~500ms, running on the ZYNQ UltraScale+.

.

10 REFERENCES

[1] C. Simmonds, *Mastering Embedded Linux Programming*: Packt Publishing Ltd, 2015.

[2] (4 October 2016). *NEON - ARM*. Available: http://www.arm.com/products/processors/technologies/neon.php

[3] (4 October 2016). *RT PREEMPT HOWTO - RTwiki*. Available: https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

[4] (4 October 2016). *The Zynq Book: About The Book*. Available: http://www.zynqbook.com/about.html

[5] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero, "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?," 2013, pp. 1-12.

[6] (4 October 2016). *OpenMP Specifications*. Available: http://openmp.org/wp/openmp-specifications/

[7] (4 September 2016). *GitBook · Writing Made Easy*. Available: https://www.gitbook.com/

[8] (4 October 2016). *UltraFast Design Methodology*. Available: https://www.xilinx.com/products/design-tools/ultrafast.html

[9] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual: For System-on-Chip Design*: Springer Publishing Company, Incorporated, 2007.

[10] A. Ahmed and W. Wolf, "Hardware/Software Interface Codesign for Embedded Systems," 2005 2005.

[11] (4 October 2016). *Android Best Practices | OpenCV*. Available: http://opencv.org/platforms/android/android-best-practices.html

[12] (4 October 2016). *Best Practices in Embedded Systems Programming*. Available: http://www.embedded.com/collections/4398825/Best-practices-in-programming

[13] (4 October 2016). *FPGARelated.com - All You Can Eat FPGA*. Available: https://www.fpgarelated.com/

[14] (4 October 2016). *EVA Blog*. Available: http://www.embedded-vision.com/industry-analysis/blog

[15] (19 September 2016). *Xcell Publications*. Available: http://www.xilinx.com/about/xcell-publications.html

[16] (4 October 2016). *EVA Technical Articles*. Available: http://www.embedded-vision.com/industry-analysis/technical-articles

[17] (4 October 2016). *Embedded Linux Experts - Free Electrons*. Available: http://free-electrons.com/doc/training/embedded-linux/

[18] (4 October 2016). *Training and Videos | Zedboard*. Available: http://zedboard.org/support/trainings-and-videos

[19] (4 October 2016). *1080p60 HD Medical Endoscope*. Available: http://www.xilinx.com/applications/medical/endoscope.html

[20] M. Wolf, *High-Performance Embedded Computing: Applications in Cyber-Physical Systems and Mobile Computing*: Newnes, 2014.

[21] E. White, *Making Embedded Systems: Design Patterns for Great Software*: " O'Reilly Media, Inc.", 2011.

[22] J. C. Russ, "The Image Processing Handbook, Sixth Edition," *CRC Press,* April 2011 2011.

[23] (4 October 2016). *Xilinx Alliance Member Design Services*. Available: http://www.xilinx.com/alliance/design-services.html#certified

[24] A. J. Barry and R. Tedrake, "Pushbroom stereo for high-speed navigation in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3046-3052.

[25] A. Paolillo, O. Desenfans, V. Svoboda, J. Goossens, and B. Rodriguez, "A New Configurable and Parallel Embedded Real-time Micro-Kernel for Multi-core platforms," *OSPERT 2015,* p. 25, 2015.

[26] B. B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Citeseer, 2011.

[27] GPU vs FPGA Performance Comparison. White paper, found: http://www.bertendsp.com/pdf/whitepaper/BWP001_GPU_vs_FPGA_Performance_Comparison_v1.0.pdf, 19.05.2016

[28] Bachrach, Jonathan, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. “Chisel: Constructing Hardware in a Scala Embedded Language.” In *Proceedings of the 49th Annual Design Automation Conference*, 1216–1225. DAC ’12. New York, NY, USA: ACM. doi:10.1145/2228360.2228584.

[29] Bacon, David F., Rodric Rabbah, and Sunil Shukla. 2013. “FPGA Programming for the Masses.” *Commun. ACM* 56 (4): 56–63. doi:10.1145/2436256.2436271.

[30] Borkar, Shekhar, and Andrew A. Chien. 2011. “The Future of Microprocessors.” *Communications of the ACM* 54 (5): 67. doi:10.1145/1941487.1941507.

[31] Canis, Andrew, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Tomasz Czajkowski, Stephen D. Brown, and Jason H. Anderson. 2013. “LegUp: An Open-Source High-Level Synthesis Tool for FPGA-Based Processor/Accelerator Systems.” *ACM Trans. Embed. Comput. Syst.* 13 (2): 24:1–24:27. doi:10.1145/2514740.

[32] Jahre, Magnus, Asbjørn Djupdal, Lester Kalms, and Ananya Muddukrishna. 2017. “D4.1: Basic Tool Chain.” TULIPP Project.

[33] Koeplinger, David, Christina Delimitrou, Raghu Prabhakar, Christos Kozyrakis, Yaqi Zhang, and Kunle Olukotun. 2016. “Automatic Generation of Efficient Accelerators for Reconfigurable Hardware.” In *Proceedings of the 43rd International Symposium on Computer Architecture*, 115–127. ISCA ’16. Piscataway, NJ, USA: IEEE Press. doi:10.1109/ISCA.2016.20.

[34] Langdal, Peder Voldnes, Magnus Jahre, and Ananya Muddukrishna. 2017. "Extending OMPT to Support Grain Graphs." In *Scaling OpenMP for Exascale Performance and Portability*, 141–55. Lecture Notes in Computer Science. Springer, Cham. doi:10.1007/978-3-319-65578-9_10.

[35] Muddukrishna, Ananya, Peter A. Jonsson, Artur Podobas, and Mats Brorsson. 2016. "Grain Graphs: OpenMP Performance Analysis Made Easy." In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 28:1–28:13. PPoPP '16. New York, NY, USA: ACM. doi:10.1145/2851141.2851156.

[36] Prabhakar, Raghu, David Koeplinger, Kevin J. Brown, HyoukJoong Lee, Christopher De Sa, Christos Kozyrakis, and Kunle Olukotun. 2016. "Generating Configurable Hardware from Parallel Patterns." In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 651–665. ASPLOS '16. New York, NY, USA: ACM. doi:10.1145/2872362.2872415.

[37] Stone, John E., David Gohara, and Guochun Shi. 2010. "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems." *Computing in Science & Engineering* 12 (3): 66–73. doi:10.1109/MCSE.2010.69.

[38] Wang, Z., B. He, W. Zhang, and S. Jiang. 2016. "A Performance Analysis Framework for Optimizing OpenCL Applications on FPGAs." In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 114–25. doi:10.1109/HPCA.2016.7446058.

[39] Zhong, G., A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar. 2017. "Design Space Exploration of FPGA-Based Accelerators with Multi-Level Parallelism." In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 1141–46. doi:10.23919/DATE.2017.7927161.

[40] Sadek, Ahmad, Ananya Muddukrishna, Lester Kalms, Asbjørn Djupdal, Ariel Podlubne, Antonio Paolillo, Diana Goehringer, and Magnus Jahre. 2018. "Supporting Utilities for Heterogeneous Embedded Image Processing Platforms (STHEM): An Overview." In *Applied Reconfigurable Computing (ARC)*.

11 Appendix: Guideline Evaluation Experts

The list of experts that can be called by the quality assurance board to assess the quality of each guideline and improve the content is described below.

In the following table, the skills are divided into the following sections:

- ***HW*** refers to expertise in hardware architecture design, chip & interfaces selection and board design.
- ***OS*** refers to operating system design and necessary low-level software libraries for the application to utilise the underlying hardware
- ***Tools*** refers to vendor tool chains, design and monitoring tools that helps application developers to implement the code on the hardware
- Application ***Algorithm*** refers to the design of the algorithms that will be used in the application to bring the required functionalities to the product
- Application ***Software*** and ***Firmware*** refers to the development of the application on the chosen platform. The application can be software written in C, C++ or any language that can execute on processors and accelerators or firmware written in VHDL, Verilog or any model-based programming tools and implemented on a FPGA.
- ***System*** refers to the design and development of the whole solution. This is necessary to ensure integration of all the pieces to a comprehensive product.

		Area of Expertise				*Application Domain*			
Name	*Affiliation*	*HW*	*OS*	*Tools*	*System*	*AI*	*Algorithm*	*Software*	*Firmware*
Magus JAHRE	NTNU	x		x	x	x		x	x
Asbjørn DJUPDAL	NTNU	x	x	x	x	x		x	x
Antonio PAOLILLO	HIPPEROS		x	x					
Ben RODRIGUEZ	HIPPEROS		x	x					
Olivier DESENFANS	HIPPEROS		x	x					
Vladimir SVOBODA	HIPPEROS		x	x					
Paul RODRIGUEZ	HIPPEROS		x	x					
Fatima KISHWAR	SUNDANCE	x			x				x
Timteo GARCIA BERTOA	SUNDANCE	x			x				x
Ben LOVELL	SUNDANCE				x				
Graeme PARKER	SUNDANCE	x			x				
Pedro MACHADO	SUNDANCE	x		x	x	x		x	x
Emilie WHEATLEY	SUNDANCE		x	x	x		x	x	
Flemming CHRISTENSEN	SUNDANCE	x			x				
Magnus PETERSON	SYNECTIVE	x		x	x		x	x	x
Carl EHERNSTAAHLE	SYNECTIVE		x	x	x		x	x	x
Roland STENHOLM	SYNECTIVE		x	x	x			x	x
Öystein STAVEN	SYNECTIVE							x	x
Svante LILJEQVIST	SYNECTIVE			x	x				x
Ahmed SADEK	TUD	x		x	x			x	x
Ariel PODLUBNE	TUD	x		x	x			x	x
Julian HAASE	TUD	x			x			x	
Diana GÖHRINGER	TUD	x		x	x			x	x
Alvin SASHALA NAIK	THALES			x			x		
Paul BRELET	THALES			x				x	x
Kevin PIDENCE	THALES							x	x
Sébastien JACQ	THALES	x						x	x
Remi BARRERE	THALES			x				x	x
Kevin EYSSARTIER	THALES							x	x
Philippe MILLET	THALES	x	x	x					
Igor TCHOUCHERNKOV	FHG	x							
Michael GRINBERG	FHB						x	x	
Boitumelo RUF	FHG				x		x	x	x

Table 3: Tulipp Expertise Panel