TULIPP

H2020-ICT-04-2015

Grant Agreement n° 688403

D1.3: Reference Platform v3

Authors

Organization	Participant
THALES	Philippe Millet
SUNDANCE	Flemming Christensen
IOSB	Michael Grinberg
IOSB	Igor Tchouchenkov
HIPPEROS	Antonio Paolillo
TUD	Lester Kalms
TUD	Ariel Podlubne
TUD	Julian Haase
SYNECTIVE	Magnus Peterson
NTNU	Björn Gottschall
NTNU	Asbjørn Djupdal
NTNU	Magnus Jahre

Reviewers

Organization	**Participant**
University of Coimbra	Jorge Lobo
University of Coimbra	Bruno Oliveira
University of Glasgow	Shufan Yang
University of L'Aquila	Giacomo Valente
University of L'Aquila	Luigi Pomante
S V National Institute of Technology (SVNIT)	Pinalkumar Engineer
Sheffield Hallam University	Kofi Appiah

Document Description

Deliverable number	D1.3
Deliverable title	Reference Platform v3
Work Package	WP1
Deliverable nature	Report
Dissemination level	PU (Public)
Contractual delivery date	31. January 2019 (M36)
Actual delivery	1. February 2019
Version	1.2

	Written by	**Approved by**
Name Signature	TULIPP consortium	

Version History

Version	Date	Description
1.0	5. December 2018	First complete version for review by the TULIPP Advisory Board.
1.1	14. January 2019	Version addressing the comments of the TULIPP Advisory Board.
1.2	1. February 2019	Final version with the latest modifications to the guidelines and addressing additional comments from the TULIPP Advisory Board.

Executive Summary

The TULIPP project focuses on high-performance, low-power, embedded image processing systems, and implementing these systems require attaining a delicate balance of conflicting requirements. In essence, these systems should provide high performance, consume minimal power, and cost next to nothing – a practically impossible combination.

Standardisation is a critical means towards improving the productivity and competitiveness of the European high-performance embedded image processing industry. A key objective of TULIPP is to define a reference platform that future standardisation effort can build upon. Generally, minimising cost favours standardisation while improving performance, power, and energy favours specialisation. Through the TULIPP *Reference Platform (TRP)* we attempt to achieve the advantages of both approaches. TRP provides a hardware platform, Operating System (OS), libraries and tools – supporting a selection of interfaces suitable for embedded image processing systems (see the below table for the complete list of TRP interfaces). To optimise for a particular application, the developer can select the most suitable interfaces and remove the others. In this way, TRP offers to satisfy both efficiency and cost requirements: The high efficiency of specialisation and the low cost provided by standardisation.

	TRP Interface	**TRP Choice**
Hardware Platform	Camera Interfaces	CPI, GigE Vision, CameraLink, HDMI
	Display interfaces	HDMI, DSI
	Communication interfaces	USB, PCIe, GigE
	Deployment interface	JTAG (IEEE 1149.1)
Operating System	Tasks implementation	C/C++
	File system	Fat32
	Network	TCP/IP, UDP/IP
	Bootloader interface	U-Boot
	Debug interface	JTAG (IEEE 1149.1)
	Parallel processing libraries	OpenMP, Posix threads
	Image processing libraries	OpenCV, OpenVX
Tool-chain	Programming language	C/C++
	Compilation and synthesis tools	GCC, LLVM, Xilinx Vivado
	Accelerator generator control	Vivado HLS annotations
	Software instrumentation interface	LLVM IR
	Object file linker	Xilinx linker
	PC sampling interface	JTAG (IEEE 1149.1)

Embedded image processing applications commonly push hardware platforms to – and sometimes beyond – their performance limits. Thus, there is no room for adding unnecessary overhead to the implementation. Thus, developers and system integrators need a way of selecting the subset of the TRP interfaces to support in a particular system. To facilitate this process, we have proposed the concept of *guidelines*. A guideline encapsulates an expert insight in a precise, context-based formulation which orients the follower towards a goal by recommending an implementation method. Within TULIPP, we have defined a number of guidelines that we hope will help future developers productively implement efficient embedded image processing applications.

Contents

List of Figures

List of Tables

1 Introduction

Image processing deals with image manipulation, transformation, and analysis. The input is two-dimensional data of a given width, height and pixel depth depending on the sensor. The wide variety of sensors, application types, and use cases makes image processing a complex and deep domain. In TULIPP, we focus on embedded applications that require high, real-time performance while satisfying stringent power and energy consumption requirements.

The exact meaning of power efficiency is heavily dependent on the context in which the term is used. For instance, we can easily find high-performance targets dedicated to data centres and clouds which dissipate hundreds of Watts. At the other end – in realm of the Internet of Things (IoT) – we can find low-power platforms that dissipate less than a Watt but deliver low performance.

There is a desert in the middle, where the systems require finding a good trade-off between sufficient performance and low-enough-power – i.e., between 1 and 25 Watt. We expect this area to develop with the ever growing needs for ADAS systems, but we might not be able to access the technology. For instance, the Xavier System on Chip (SoC) from NVIDIA is dedicated to the automotive market. This market is so large that NVIDIA has chosen to focus on major customers and only sells the SoC with a strong support for several million euros. The same situation can be observed for processors dedicated to smart phones – chips are sold in big quantities only.

For smaller series of products, developers need to select platforms that meet processing, price, and power consumption requirements. 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-offs for their system. For these systems, we provide the TULIPP Reference Platform (TRP) which is the key contribution of this report. The TRP is based upon a deep understanding of the available technology and significant analysis on how image processing systems can be efficiently implemented. Through the TULIPP use cases, we show that a set of diverse image processing applications can be efficiently implemented on our TRP instance.

1.1 Embedded Image Processing Design Trade-offs

Ideally, embedded image processing systems should have high performance, dissipate minimal power, and cost as little as possible – a classic case of conflicting objectives. Thus, implementing an efficient image processing application requires carefully trading off different alternatives.

In summary, the following aspects need to be considered:

- **Power and energy requirements:** Embedded systems are often limited by battery life, but even when the product has access to the electrical grid it can be limited by the thermal dissipation within the heat sink, the cabinet or the packaging. Therefore, energy consumption, power dissipation, and thermal issues commonly place restrictions on the implementation of image processing systems.

- **Performance:** Image processing applications tend to require more and more performance to deal with the large data sets provided by newer sensors. This creates a push for more powerful compute platforms – as this makes it easier for software developers to meet performance requirements.
- **Non-Recurrent Costs (NRCs):** NRCs are costs incurred during the development of the product. Higher NRCs might not be a problem when you develop a high-volume product, but for low-volume products care must be taken to reduce NRCs since they may significantly increase the price of the product.
- **Recurrent Costs (RC):** RCs are costs incurred during the production phase of the product and strongly depends on the choices made at design time – as more expensive components means higher prices. Higher RCs will always impact the final product price, but unlike the NRC you have to take greater care with high-volume products as each cent saved on a product can lead to millions gained in revenue.

A trade-off does not mean the hardware designer has to find a solution that matches all constraints. Commonly, the designer needs to discuss the constraints and move the thresholds. If all the functionalities cannot be implemented, the designer can see 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 by 1% to 2% might in some case reduce the load by several tens of percentage points and allow for using smaller and cheaper components.

1.2 The TULIPP Reference Platform (TRP)

Generally, the cost requirements of embedded image processing applications favour standardisation while the performance, power, energy, and thermal requirements favour specialisation. Through the TULIPP Reference Platform (TRP) we attempt to achieve the advantages of both approaches. The TRP provides a hardware platform, Operating System (OS), libraries and tools – supporting a selection of interfaces suitable for embedded image processing systems. To optimise for a particular application, the developer can select the most suitable interfaces and remove the others. The TRP also contains an FPGA fabric which enable implementing application-specific accelerators. In this way, TRP offers to satisfy both performance and cost requirements: The high efficiency of specialisation and the low cost provided by standardisation.

To help the developer choose which interfaces to focus on, we provide a rich set of guidelines. A guideline encapsulates an expert insight in a precise, context-based formulation which orients the follower towards a goal by recommending an implementation method.

Reaching the performance potential of the TRP requires adapting an image processing algorithm to leverage the components of the platform as well as making a number of application-dependent trade-offs. To efficiently support this procedure, we propose the Generic Development Process (GDP) [20] (see Figure 1). 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.

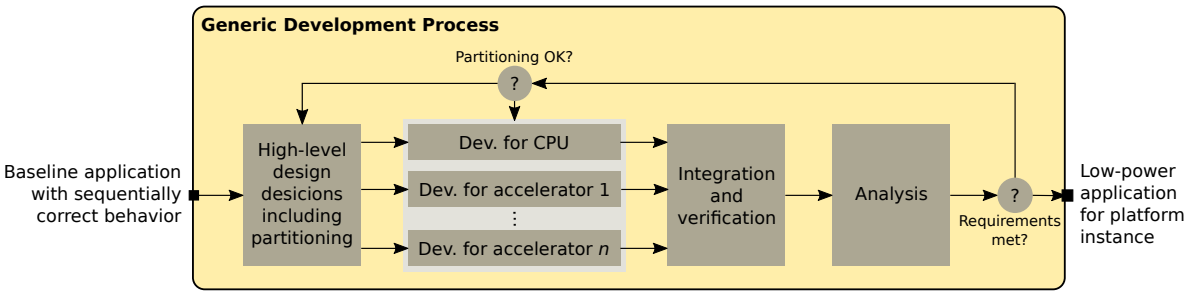


Figure 1: The Generic Development Process (GDP)

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. This is the initial development step for most image processing systems – ensuring that all the functions of the system are fully understood. Although this is a critical step, substantial effort is commonly needed to move the system onto an embedded platform.

GDP starts when the application runs correctly on the embedded platform. The first task is high-level partitioning which decides 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 software support. This will result in very low developer productivity since GDP is reduced to a time-consuming trial-and-error process. Furthermore, the developer will spend significant time implementing low-level management routines and schedulers as well as support code for identifying performance problems. To avoid these issues, the TRP includes an Real-Time Operating System (RTOS), image processing libraries, as well as extensive development and performance analysis tools. This enables the developer to focus on developing the image processing system – and leverage the capabilities of TRP's support software when needed.

1.3 Structure of the Deliverable

This deliverable describes the final version of the TULIPP Reference Platform (TRP), and consists of four parts:

- Section 2 describes typical embedded image processing applications in detail and sets the context in which the TRP will be used.
- Section 3, 4, and 5 describe hardware, Operating System (OS), and tools selection, respectively. The sections list and discuss the most important implementation alternatives and the interface selection procedures we used in TULIPP.

- Section 6 presents the interfaces of the TRP, and Section 7 introduces the guidelines concept we provide for selecting the interfaces for TRP instances that are optimised for a given application.
- Section 8 concludes the deliverable and lists the interfaces of the TRP. In addition, it proposes future standardisation efforts for key TRP interfaces which the TULIPP consortium believes could benefit from standardisation.

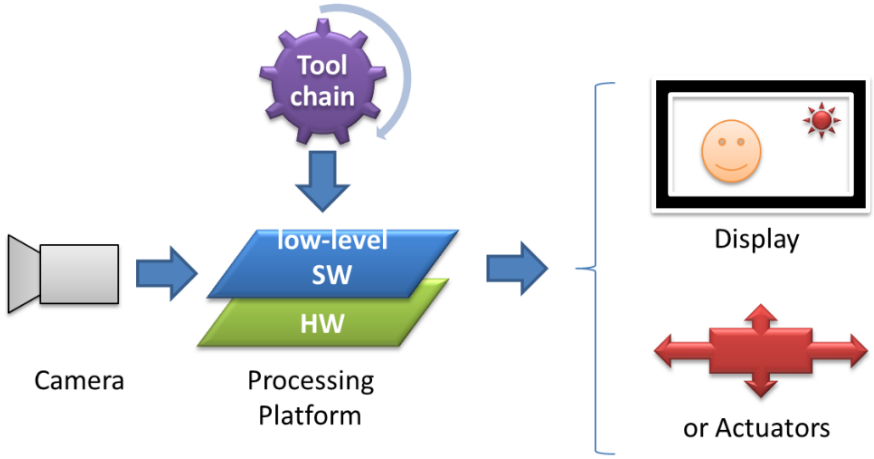


Figure 2: The typical usage for an embedded image processing platform

2 Embedded Computing Challenges

Embedded computing refers to a computing solution that performs a dedicated function within a larger mechanical or electrical system. The larger system is often mobile, commonly process sensor data, and may need to perform computation within given deadlines (i.e., real-time constraints). Embedded systems bring functions such as control and automation to devices in common use today – e.g., mobile phones, washing machines, cameras, peacemakers, TVs, and alarm clocks. A recent study found that 98% of all manufactured microprocessors are components of embedded systems [5].

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 a 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 – e.g., the Google GPS application on Android mobile phones uses the network servers to compute the best way to reach a given destination.

2.1 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 (see Figure 2).

Such an image-processing platform is dedicated to processing images from one or more sensors (e.g., a camera or camera-like device). Through a dedicated algorithm, higher-level 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 each 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 not to 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.

To achieve this, a holistic view of the system is required to obtain 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 them. This design step highly benefit from an operating system since it for instance transparently maps and schedules compute jobs onto hardware compute devices.

With a power-aware tool chain, the application designer could check, for each mapping of the application tasks on the hardware resources, the impact on power consumption. The designer can thus schedule the processing chain to optimise 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 optimised implementation of necessary set of common image processing libraries and allows a seamless scheduling of the application on the hardware chips.

2.2 Medical Challenges

As defined by the physicians, Medicine is an art based on science. Doctors have to diagnose, to make prognosis, and to make decisions based partly on protocols and scientific examination of the patient. The difficulties they face are mostly to be able to understand what is 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. Technological improvements enable medical staff to benefit from more accurate measurements 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 key mechanisms to improve diagnostic accuracy, reduce the time spent to cure patients, or to increase the level of control while administering the cure. 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.

Dedicated to X-ray instruments, TULIPP addresses a significant part of the market share, namely the mobile C-arm, which is a perfect example of a medical system that improves surgeon efficiency. This device shows the doctor a real-time view from inside the body of the patient during the operation, allowing for small incisions instead of wide-cuts and for more accurate targeting the desired region. As a result, a much faster recovery of the patient and reduction of nosocomial diseases risks are achieved. The drawback of this technique is the radiation dose, which is 30 times higher than what we receive from our natural surroundings each day. Such high dose is received not only by the patient but also by the medical staff doing such interventions all day long, several days a week.

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

It is possible to divide the radiation by 4 and restore the original quality of the picture thanks to specific noise reduction algorithms running on high-end PCs. Unfortunately, in such a confined environment as an operating room, crowded with staff and equipment, when size and mobility mater, this is not convenient.

By bringing the computing power of a standard Intel core-i7 PC to hardware with the size of a smartphone, TULIPP makes it possible to lower the radiation dose while maintaining the picture quality. To achieve this goal, a holistic view of the system is required to get the best power-efficiency from a necessarily highly heterogeneous hardware.

2.3 UAV and Drone Challenges

The term UAV (unmanned Aerial Vehicles) refers to any flying aircraft without humans on board. In recent years, the usage of UAVs in different application fields has increased significantly. Currently, most important markets for UAVs are aerial photogrammetry, panoramic

(a) (b) (c)

Figure 3: Different obstacles in typical environments of UAVs

photography, precision farming, surveillance and reconnaissance. Further application fields are rescue, law enforcement, 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 of their capabilities; perception and intelligent evaluation of the environment make many new applications possible.

In most cases, UAVs are carrying sensor payload that allows to accomplish the respective task. Even though we are used to hear about autonomous drones, most of the current systems are still remotely piloted by humans. A human on the ground has to permanently monitor both the payload of the drone in order to successfully accomplish the desired mission (e.g., to capture the desired data) and the drone flight in order to avoid collisions with obstacles.

The simultaneous operation of the UAV and of the sensor payload is a quite challenging task. Mistakes may be fatal either with regard to the mission success or — much worse — with regard to the mission safety. Besides, there might be a limitation of the UAV operation area due to the need for a constantly available communication link between the UAV and the remote control station.

Most of the small UAVs fly with velocities up to 60 km/h (17 m/s), but some of them can fly with velocities up to 180 km/h (50 m/s) and even more. The stopping distance is strongly dependent on the velocity, the type of the UAV and the payload. With the regular velocity of about 30 km/h of a common type of “multicopter”, the stopping distance is less than 5 m, but with more extreme UAVs it can be up to 50 m. The size of a dangerous obstacle is usually over 5 cm (Figure 3a), but in some cases obstacles with sizes of as less 2 mm, more difficult to detect, can also be dangerous (Figure 3b and 3c).

As always, each technology comes along with drawbacks and potential for unintended abuse and this is in particular true for UAVs. With the growing number of UAVs also the number of crashes due to malfunction or maloperation increases. In worst cases it might cause damages to goods, infrastructure, and – even worse – people. 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.

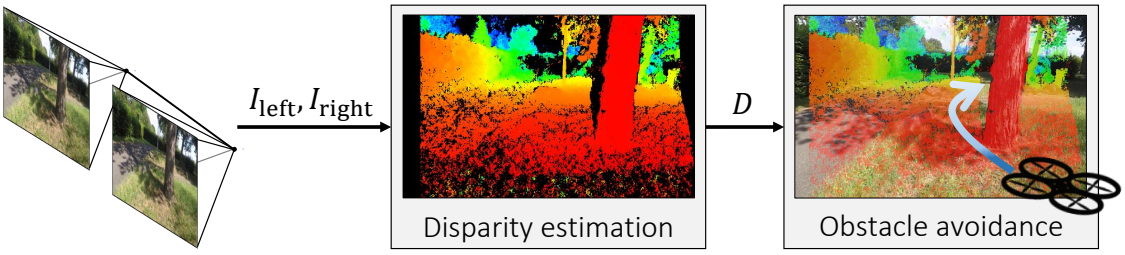


Figure 4: Setup of the “UAV use case”

A major improvement could be made if UAVs were capable of autonomous navigation or at least if they had an obstacle detection and avoidance capability. Such capability can be achieved by means of additional sensors, such as ultrasonic sensors, radars, laser scanners, or video cameras, that are monitoring the UAV surroundings. However, the ultrasonic sensors have a very limited range, the radar sensors might “overlook” non-metal objects and the laser scanners are heavy and energy-intensive and thus not well suited for applications with UAVs that suffer from tight weight and power constraints. Hence, our work within TULIPP focused on obstacle avoidance based on a lightweight stereo camera setup with cameras orientated in the direction of flight. The principle idea is shown in Figure 4

Unfortunately, more intelligence means more computing power which, in turn, means more weight and energy consumption. These are, however, very limited for small UAVs. The UAV use case in TULIPP deals with low-power stereo image processing for UAV navigation. The disparity maps, which are computed from the camera images, are used for obstacle localisation and collision avoidance. While implemented for UAVs, the technology is easily portable for the application on other vehicles and particularly cars.

The collision avoidance system have to support the following functions:

1. Synchronous image acquisition from two video cameras
2. Image rectification, i.e., transformation of the images in a way that pixels corresponding to the same world point come to lie in the same image row of both images
3. Computation of dense disparity maps by matching pixels in both images
4. Detection of objects which are in the current flight path of the UAV and can cause a collision
5. Collision avoidance function, which computes the shortest route around the detected obstacles taking into account inaccuracies of the 3D model and possible deviations 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. The TULIPP solution ensures this as well as appropriately balancing weight, performance and power consumption [37].

2.4 ADAS Challenges

Advanced driver-assistance systems (ADAS) – which help the driver to focus on what is important on the road – are developing at a very fast pace. Sustained by the 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 is drained by the chips and this becomes even worse when more computation and more powerful processors are required. A second challenge is the number of sensors are 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 make the right decision.

This second challenge also comes with a price. The technology is 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 since this is where the highest return on investment 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 [41]. 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 with 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, 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 classifiers or Convolutional Neural Networks is one of the most important challenges of the coming 5 years.

2.5 Neural Network Challenges

The last 5 years, with the development of Deep Neural Networks (DNNs), neural networks became the best way to analyse information with a high quality of results and low error rates. Even though many kind of neural networks have been created, in the domain of image and video analysis, the main basic computation bloc is a convolution. These neural networks are therefore called Convolutional Neural Networks (CNNs).

The images, or the video-frames, are processed through several layers of the network, each layer being in charge of extracting meaningful details from the images.

Figure 5 shows a schematic view of a convolutional neural network of five layers. The neurons are schematised with a black circle in the layer. The connections between the layers are schematised with an horizontal arrow. The first layer, the input layer, is in charge of extracting the pixels from the image and may apply a first convolution or a filter to the image. The last layer is the output layer. It is in charge of collecting the information from the neural network. The internal layers between the input and output layers are in charge of splitting and partly characterising the image through a list of features and then combining the features to identify and classify the information extracted from the image. Each neuron on one layer provides a set of outputs that are sent to other neurons on the next layer. Each of such a connection between two neurons from two layers is weighted. The set of weights of the connections between the neurons and the weights of the convolutions are the parameters of the neural network.

While the performance of these networks is getting higher at each generation, it is difficult to make them run in real-time on processors targeted for embedded products [46]. There are two main constraints to achieve this goal: (1) the amount of computation to process the convolutions is too high to be performed at real-time and (2) the available memory on embedded processing platforms is too small to fit both the data and the parameters of the network.

Neural networks are, however, very static. Once a network is designed and parametrised

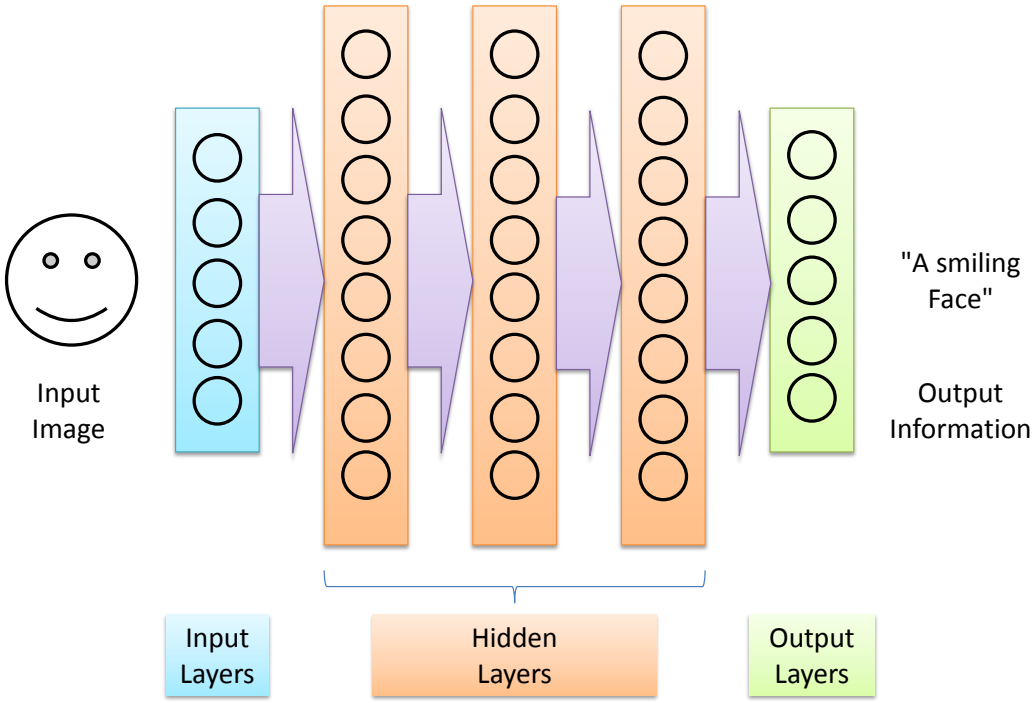


Figure 5: Basic structure of a Convolutional Neural Network - CNN

through a learning phase, its implementation and the data path between the layers is known and predictable. This characteristic makes it possible for hardware designers to finetune architectures to efficiently execute CNNs. Dedicated processors with accelerators for convolutions are necessary together with highly efficient memory access to high quantity of memory. Due to the predictability of the CNNs, the memory might be highly hierarchical, automatic code generation tools would then be helpful to schedule the execution of the network and the flow of data between the memories and the processing units of the architecture.

2.6 Challenges Conclusion

The challenges might be summarised as more and more processing is required to process more and more sensors with always higher quality, bigger picture sizes and add more and more the capability to interpret the images (see Table 1).

Table 1: Energy-efficient embedded image processing applications challenges.

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 form 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, etc.).
Energy and Power	The energy consumption should ideally remain constant. While this might not be possible, it must be managed as more energy means bigger batteries with higher costs and weight. Meeting the power-budget means much higher processing-efficiency is required.
Development Costs	Development costs 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.

3 Hardware Platforms

The hardware platform refers to the underlying hardware on which the software runs. While the software comprises the operating system, the libraries and the final application, the hardware platform is made of the board, the processor 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.

Being embedded, the hardware part of the system has to fight to keep energy consumption and heat dissipation within requirements. 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. Since the hardware has to be integrated into a system with other devices, it must take into account the interfaces of those devices as well as the throughput and latency requirements of the application.

3.1 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.

3.2 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 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, you can collect information that will be a first filter for the chips:

- The thermal design power (TDP) which tells you the maximum power you have to evacuate off the chip through cooling systems. In the medical use case for instance, the maximal power dissipation is 8 W. The board will have to have a global TDP less than 8 W, which implies the sum of the TDP of the components on the board must be less than 8 W.

- The operating temperature tells you 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 is 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 you multiply it with time, it gives you 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 since bigger batteries mean more weight. If the solution is too heavy, the UAV might not even take off.
- The interfaces indicate how easy it will be to integrate the SoC into the overall system. But the interfaces you need for an application must also be supported by the operating system, tool-chain and libraries of interest (like OpenCV), and this information is not always available – especially for new chips.
- The programming language and APIs tell how easy you will find programmers that can start writing code for the chip. But this will be be enough to tell you you how easy the programmers will be able to develop an application on the chip while it depends from other factors like debug environment, ability to monitor the activity of the chip, means to find deadlocks, complexity of the hardware, availability of the documentation, etc.

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 ability: Whether real data sent by sensor can be processed so quickly that no significant degradation of system functionality is expected
- Ease of programming: When the development team starts to develop on the target, they can evaluate if it is hard to manage or not. The results of the evaluation on the target will strongly depend on the way it is implemented.

For the right choice of hardware, the tasks to be solved must be thoroughly analysed – and not just theoretically. In the following, selection of hardware is described using as practical example the TULIPP reference platform instances.

3.3 Selecting the TULIPP Hardware Platform

Within TULIPP, we can only develop a single hardware platform. The reason is that the OS, tool-chain, vendor tools and use cases all need to be adapted to the specifics of the platform to achieve high performance. Thus, hardware platform selection has a project-wide impact.

In order to create a solid basis for comparing different platforms, we conducted a market survey in which we examined the platforms provided by key vendors such as Intel, NVIDIA, Qualcomm, Samsung, AMD, and Xilinx and many others. Interestingly, many of vendors could not offer platforms that meet the requirements of the TULIPP project. Many available

	Power	Performance	Performance per Watt	Interfaces	Release Year
Movidius (Myriad2)	1,2 W	150 GFlops		12 Lanes MIPI, 3xI2C, SPI, 3xI2S ,GPIO, PWM, USB3.0, 2-Slot SDIO, 1xUART, 1xGbE, parallel video I/O	2016
Nvidia Tegra K1	11 W	326 GFlops		Input: CSI (4x4x1); USB 2.0/3.0, I2C, 2x DSI, PCIe, GPIOs, GbE, HDMI 4K	2014
Nvidia Tegra X1	11 W	1024 GFlops		2x DSI, eDP 1.4 / DP 1.2 / HDMI 2.0, UART, SPI, I2C, I2S, GPIOs, USB 2.0/3.0, PCIe 2.0, GbE	2015
Nvidia Tegra X2	15W	1500 GFlops	100 GFlops/W	2x DSI, eDP 1.4 / 2x DP 1.2 / HDMI 2.0, CAN, UART, SPI, I2C, I2S, GPIOs, USB 2.0/3.0, PCIe 2.0, GbE	2017
KeyStone II (66AK2H14)	14 W	198 GFlops		10-GbE, 2xPCIe 2.0, USB 3.0, 3xI2C, 3xSPI, 2xUART, EMIF16,…	2014
Sitara (AM5728)	6,5 W	10500 DMIPs		2 PRU-ICSS, QSPI, 2xPCIe 2.0, GPIO, USB 3.0, 2xCAN, 1xHDMI out, 2xGbE,…	2015
Snapdragon 820	?	500 GFlops		USB 3.0,Bluetooth 4.1, 4K Ultra HD, UFS 2.0, eMMC 5.1, SD 3.0	2015
Exynos 8890	?	265 GFlops		UFS 2.0, eMMC 5.1, 4K Ultra HD	2016
Atom X5 (Z8500)	2 W	115 GFlops		USB 3.0, 1xPCIe 2.0, HDMI 1.4	2015
Atom X7 (Z8700)	2 W	153 GFlops		3xUSB 3.0, 2xPCIe 2.0, HDMI 1.4	2015
Apollo Lake (N4700)	6 W	230 GFlops		6 x PCIe 2.0, HDMI 1.4b, 6x USB 3.0, 2x SATA 3, I2C, SPI,…	2016
Kalray MPPA2-256	20 W	1500 GFlops	75 GFlops/W	2x 40GbE and PCIe x16 3.0	2016
Adapteva Epiphany-IV	2 W	140 GFlops	70 GFlops/W	2xeLink, 24 GPIO	2013
AMD G-Series (I/J)	8-15 W	564 GFlops		PCIe 3.0 1x4, PCIe 2/3 4x1, 2xUSB3.0, 2xUSB2.0, 2xSATA 2.0/3.0, 2xHDMI 2.0,…	2016
Stratix 10 (GX/SX 400)	12,5 W	1000 GFlops	80 GFlops/W	PCIe 3.0, 2xUSB 2.0, 3xGbE, 2xUART, 4xSPI , 5xI2C, 1x eMMC 4.5,…	2016
Zynq 7000	1-20 W	60-1560 GFlops	72 GFlops/W	2xUSB2.0, 2xGbE, 2xQSPI, 2xI2C, 2xCAN 2.0, 2xUART,…	2013
Zynq UltraScale+	Up to 24W	2x better	130 GFlops/W	PCIe 2.1, 2xUSB3.0, sATA 3.1, 4xGbE, 2x eMMC 4.51, 2xQSPI, 2xI2C, 2xCAN 2.0,…	2016

Figure 6: TULIPP platform candidates after pruning the results of the market survey. The listed platforms offer high performance at an acceptable power consumption.

high-performance processors would deliver enough computing resources but consume over 40 W, and even hundreds of watts for the top-level performance. On the other hand, a lot of work has been done on the Internet of Things (IoT) side, where devices consuming about 1 W can be found with an appropriate performance per Watt. However, such devices have very limited memory capacities that are not sufficient for coping with the amounts of data that are to be handled within the targeted image processing applications. Only a few platforms where able to meet the performance requirements of our use case applications at a relatively low typical power consumption (less than 25 W). Power consumption is fundamentally linked to clock frequency [16] – and thereby performance. Thus, there is no free lunch – designer need to accept a somewhat higher power consumption to meet the performance requirements of image processing applications.

Figure 6 summarises the results of our market survey. We focused on newer System on Chips (SoCs) and Multi-Processor System on Chips (MPSoCs) since these tend to offer higher performance per Watt. The key reason is that they include more heterogeneity which gives developers a range of different processing units – with different performance and power characteristics. These units can be leveraged to achieve highly energy efficient image processing systems.

In the second step, we developed component selection algorithm. We followed the approach recommended by K. C. Laudon [21] which includes the following steps:

1. *Requirement analysis:* The first step is to understand the user's requirements within the framework and the environment in which the system will be installed.

	Sizes, mm	Weight, g	Interfaces		Resolution		Power Consumption	Progr. Language
			Input	**Output**	**minimal**	**optimal**		
Medical Imaging			PCIe 1x 2.0 minimum	1x Gigabit Ethernet	1024x1024	1344x1344	< 10 W, better 5W	C/C++, CUDA, OpenCL
Automotive			Camera Link	Ethernet…	640x480	1024x512	Few watts	C/C++, CUDA, OpenCL
UAV	120 x 120	< 300	2 x Camera Link	USB,CAN, Ethernet	376x240	640x480	< 10 W, better 5W	C/C++, CUDA, OpenCL, OpenMP

Figure 7: Summary of the key requirements of the TULIPP use cases.

2. *System specification*: The system specification must be clearly defined and reflect characteristics of the chosen application.
3. *Evaluation and validation*: In this phase, we rank the candidate platforms and determine how they perform relative to the user's requirements. We consider metrics such as price, availability, and technical support.
4. *Vendor selection:* This step determines which vendor and platform has the best combination of reputation, reliability, service record, training, delivery time, and lease/finance terms.
5. *Post installation review:* This step verifies that the user's requirements were fulfilled.

In the following sections, we will describe how we selected the components based of this approach.

3.3.1 Requirements Analysis

The requirements can take into account a diverse set of characteristics including, performance (e.g., speed, capacity, throughput, image resolution), connectivity, scalability, compatibility, energy efficiency, power consumption (e.g., maximum, typical), power-efficiency, physical factors (e.g., size, weight, temperature), availability, reliability, support, programming interface, and cost (e.g., platform cost, development cost, licensing fees). In TULIPP, we focus on size, weight, connectivity, image resolution, power consumption, and programming interface needed for use cases (see Figure 7).

The different TULIPP use cases put different weight on each requirement. Thus, the key task is to select a platform that meets the requirements of all use cases. Since our use cases are drawn from different domains, it is likely that the platform is suitable for a range of image processing applications.

Other requirements on components and boards are not so easy to define, because algorithms and functions necessary for the different use cases could need different components to be best implemented. In Figure 8, the main functions needed for TULIPP use case implementations are shown.

	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
Medical Imaging	Image cleaning	Pre-filtering	Multiscale edge & contrast filtering	Post-filtering		
Automotive	Search ranges configuration	Detections computation at a specific scale	Performing NMS on detected bounding boxes	Detections collection		
UAV	Image rectification	Image preprocessing	Correspondence search	Costs optimization	Disparity computation	Depth analysis

Figure 8: Main functions needed for TULIPP use cases implementation.

We do not necessarily know which components are the best choice for each of the functions in Figure 8. In the literature, some common rules to compare and select the more suitable components are available [6, 52]. Other researchers compare implementations of concrete algorithms on concrete hardware [4, 14, 44]. It is important to mention that in many cases the implementation of the same algorithms is impossible for example on GPUs and FPGAs. In particular, FPGAs tend to require specialised algorithms [43], and some functions could need many components (e.g., both CPUs and an FPGA) to be best implemented. In addition, the TULIPP Reference Platform (TRP) must be usable for other use cases of low-power, real-time image processing to enable a living ecosystem.

In conclusion, the most flexible solution is to select an up-to-date MPSoC (to achieve high energy efficiency) containing multiple CPUs, an FPGA substrate and GPUs. In this case, each function of the different use cases could be implemented using the most suitable hardware component. That also ensures other important advantages for accelerating design and re-design of program code, because software (re)development in most practical cases require much higher effort than hardware selection. However, we must ensure that the OS & drivers, tool-chain and libraries are compatible with the selected platform — as no user code can be ported without these components. To support complex algorithms and multiple video sources in parallel, the solution must support connection of multiple MPSoCs.

3.3.2 System Specification

The system specification for TULIPP use cases is based on the above analysis:

- Performance (speed, capacity, throughput)
  - Throughput: 940 MBit/s
  - Speed (MFLOPs): depending on use case
- Connectivity
  - Video Input: GbE, CameraLink
  - Video Output: HDMI
  - Control output (UAV use case): USB or CAN
  - Connection of multiple MPSoCs: PCIe 2.0

- Scalability: different types of MPSoCs (computing power/energy consumption) should be available
- Energy efficiency:
  - maximum power consumption less than 25 W
  - high efficiency (for example, MFLOPs/Watt),
  - Modern power saving capabilities (DVFS, dynamically switch off unused IOs and components etc.)
- Physical compatibility
  - Working temperatures: –65 -+120
  - Small sizes and weight (boards)

The specification contains many more or less “hard” requirements, but it does not select the hardware components definitely in all cases. In complex cases, additional parameters must be analysed to make a decision: costs, availability, reliability, support, etc.

3.3.3 Evaluation and Validation

In this step, we analysed platforms listed in Figure 6 based on the above mentioned requirements and system specification. The selection was done with weighted-sum or scalarization method [50]. For that, the relevant parameters of the components from Figure 6 were graded with numbers (scores). NVIDIA Tegra X2, Altera Stratix 10 (400), and the Xilinx Zynq UltraScale+ have turned out in this way to be the best suitable hardware platforms for the TULIPP use cases and are analysed here in more detail.

NVIDIA Tegra X2: The Tegra X2 is a mobile integrated graphics solution by NVIDIA. It is built in a 16 nm process, and based on the GP10B graphics processor. The device features 256 shading units, 16 texture mapping units and 16 Raster Operations (ROP) accelerators. The GPU is operating at a frequency of 854 MHz, which can be boosted up to 1465 MHz [29]. It draws up to 15 Watt.

Figure 9 shows the structure of NVIDIA Tegra Pascal GPU. The main features of NVIDIA Tegra X2 are:

- CPU: NVIDIA Denver2 ARMv8 (64-bit) dual-core + ARMv8 ARM Cortex-A57 quad-core (64-bit), clock up to 1465 MHz
- RAM: up to 8 GB LPDDR4
- GPU: Pascal-based, 256 CUDA cores
- Performance: up to 1,500 GFLOPS
- TSMC 16 nm FinFET process
- TDP: 7.5-15 W
- Support DirectX 12.0, OpenGL 4.6, OpenCL 1.2, Vulkan 1.1.82, CUDA 6.2

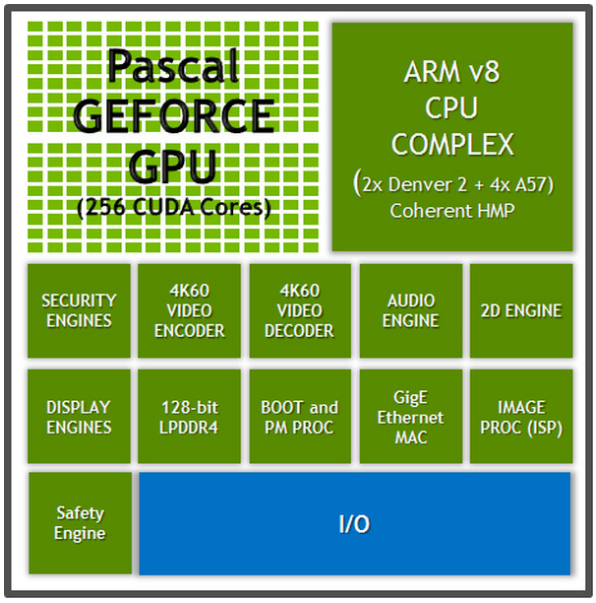


Figure 9: NVIDIA Tegra GPU

Altera Stratix 10 SoC: Altera Stratix 10 SoCs are manufactured with Intel 14 nm FinFET process technology. They feature the new HyperFlex core architecture. It uses 64 bit quad-core ARM Cortex-A53 Hard Processor System (HPS) as shown in Figure 10 [17].

The main features of Stratix 10 SoCs are:

- Quad-core 64-bit (ARMv8) ARM Cortex-A53 operating at up to 1.5 GHz HPS
- Vector floating-point unit single and double precision, ARM Neon media processing engine for each processor
- 256 KB on-chip RAM
- 8-channel direct memory access
- Intel Hyperflex FPGA Architecture delivering 2X the core performance gains
- FPGA fabric with 5.5 M logic elements in a monolithic implementation
- Up to 10 TFLOPS of single-precision floating-point DSP performance
- Up to 144 transceivers with data rates up to 30 Gbps
- Over 2.5 Tbps bandwidth for serial memory with support for Hybrid Memory Cube
- Over 2.3 Tbps bandwidth for parallel memory interfaces with support for DDR4 SDRAM at 2,666 Mbps
- PCIe Gen3 DMA to DDR4 SDRAM
- Integrated Secure Device Manager (SDM)
- Intel SoC FPGA Embedded Development Suite (EDS) featuring the ARM Development Studio 5

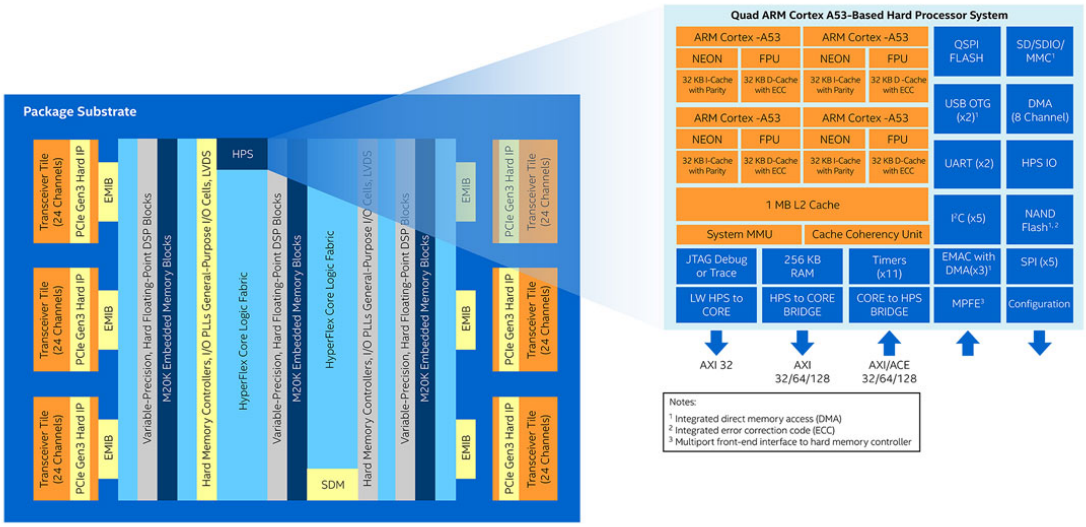


Figure 10: Structure of Altera Stratix 10 SoC

- C-based design entry using the Intel FPGA SDK for OpenCL
- Board support packages for popular operating system including Linux, VxWorks, uC/OS-II and uC/OS-III, etc.

Xilinx Zynq MPSoC Ultrascale+: The newest Zynq product family, the Multi-Processor System on Chip (MPSoC) Ultrascale+ supports many different interfaces – some of which can be used both externally and internally. The UltraScale+ enables dual-voltage operation, enabling to choose either the highest absolute performance or the highest performance-per-watt [53]. The MPSoC has all modern power saving capabilities (supports DVFS as well as dynamically switching off unused I/Os and components). For power management, there are four independently controllable power domains: the Programmable Logic (PL) plus three within the Processing System (PS) (full power, lower power, and battery power domains).

Figure 11 shows the structure of UltraScale+ MPSoC. The components of the MPSoC are interconnected and connected to the FinFET+ Programmable Logic (PL) through a multi-layered ARM AMBA AXI non-blocking interconnection that supports multiple, simultaneous master-slave transactions. Traffic through the interconnection can be regulated by the quality of service (QoS) block. Twelve dedicated AXI 32-bit, 64-bit, or 128-bit ports connect the PL to high-speed interconnection and DDR in the PS via a FIFO interface.

The APU communicates to the rest of the system through 128-bit AXI coherent extension (ACE) port via Cache Coherent Interconnect (CCI) block, using the System Memory Management Unit (SMMU). The APU is also connected to the PL through the 128-bit accelerator coherency port (ACP), providing a low latency coherent port for accelerators in the PL. To

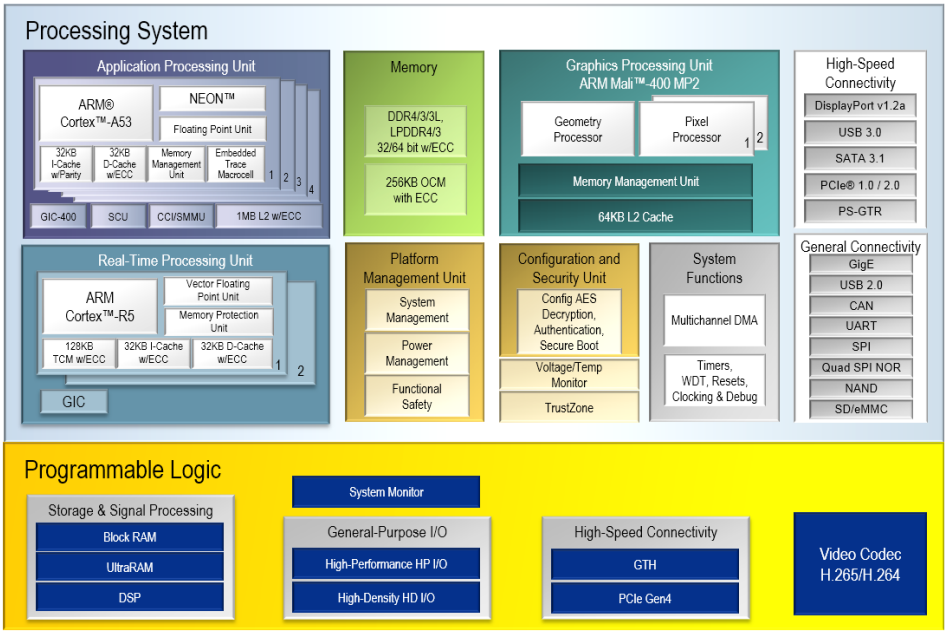


Figure 11: Xilinx Zynq UltraScale+ MPSoC [53]

support real-time debug and trace, each core also has an Embedded Trace Macrocell (ETM) that communicates with the ARM CoreSight Debug System. The Real-Time Processing Unit (RPU) contains a dual-core ARM Cortex-R5 (32-bit real-time processor based on ARM-v7R architecture). The RPU can operate in either split or lock-step mode and communicates with PL and with the rest of the PS via the 128-bit AXI-4 ports connected to the low power domain switch.

The dedicated ARM Mali-400 MP2 Graphics Processing Unit (GPU) supports 2D and 3D graphics acceleration up to 1080p resolution. The Mali-400 supports OpenGL ES 1.1 and 2.0 for 3D graphics and Open VG 1.1 standards for 2D vector graphics. It has a geometry processor (GP) and 2 pixel processors to perform tile rendering operations in parallel. It is fully autonomous, enabling maximum parallelization between APU and GPU and has built-in hardware texture decompression, supports alpha blending of multiple layers in hardware without additional bandwidth consumption etc. The GPU supports clock and power gating for each GP, pixel processors, and L2-cache. During power gating, GPU does not consume any static or dynamic power; during clock gating, it only consumes static power.

The MPSoC can interface many types of external memories through dedicated memory controllers. The dynamic memory controller supports DDR3, DDR3L, DDR4, LPDDR3, and LPDDR4 memories. The multi-protocol DDR memory controller can be configured to access a 2GB address space in 32-bit addressing mode and up to 32GB in 64-bit addressing mode using a single or dual rank configuration of 8-bit, 16-bit, or 32-bit DRAM memories. Both 32-bit and 64-bit bus access modes are protected by ECC using extra bits.

The SD/eMMC controller supports 1 and 4 bit data interfaces at low, default, high-speed, and ultra-high-speed (UHS) clock rates. This controller also supports 1-, 4-, or 8-bit-wide eMMC interfaces that are compliant to the eMMC 4.51 specification. eMMC is one of the

primary boot and configuration modes for Zynq UltraScale+ MPSoCs and supports boot from managed NAND devices. The controller has a built-in DMA for enhanced performance.

The Quad-SPI controller is one of the primary boot and configuration devices. It supports 4-byte and 3-byte addressing modes. In both addressing modes, single, dual-stacked, and dual-parallel configurations are supported. Single mode supports a quad serial NOR flash memory, while in double stacked and doubles parallel modes, it supports two quad serial NOR flash memories. The configuration rules of the internal interfaces are depending on the requirements and the usage of the board as far as on the types and the amount of components (e.g.DDR4 memory or SD card).

The main features of the Xilinx Zynq UltraScale+ MPSoC are:

- A quad-core, 64-bit ARM Cortex-A53 processor cluster with L1 and L2 caches and ECC with the ARMv8 architecture supports hardware virtualization
- Cache-coherent interconnect that provides two-way coherency between the PS and PL
- An SMMU (system memory-management unit) for PS and PL memory virtualization
- Dual-core ARM Cortex-R5F processor cluster (with floating-point extensions) capable of running in lockstep or split (independent) modes with caches and ECC memory, power-gated as a pair
- ARM Mali-400 MP2 GPU for 2D/3D graphics display and a DisplayPort interface that supports one or two streams of 4Kp30 video
- 265/H.264 video encoder/decoder supporting 4Kp60 at 10-bits/pixel
- Configuration and power-management units based on triple-redundant processors
- A DDR SDRAM controller that supports DDR3/4 and LPDDR3/4 SDRAMs with ECC. SDRAM can be shared between the Zynq UltraScale+ MPSoC PS and PL
- Hardened peripherals including Gigabit Ethernet, USB2/3, SATA3, SPI, I2C, CAN bus, PCIe, UART and Flash memory controllers (QSPI-NOR, SD, eMMC, ONFI NAND)

The Ultrascale+ MPSoCs are extremely versatile devices capable of delivering high-performance using specialised, customisable and optimised intellectual property cores (IPs).

Platform comparison: The most relevant TULIPP use cases parameters are shown in Figure 12. All platforms support all interfaces needed for the TULIPP use cases, and they have the best computing power to energy consumption values of all analysed platforms.

In our evaluation, the Zynq UltraScale+ MPSoC obtains the highest score. This is mainly due to its versatility since the TULIPP use cases have quite different requirements. Moreover, it is the only device containing both an FPGA and a GPU. This can be important for the efficient implementation of algorithms needed both technologies because can prevent bottlenecks in some cases – even if the performance of the ARM Mali-400 MP2 GPU is inferior to the Tegra X2.

	Tegra X2	Stratix 10 SoC (400)	Zynq UltraScale+ MPSoC (EV)
Processing System			
Application Processing Unit	2 "Denver 2" + 4 ARM Cortex-A57 up to 1.4 Ghz	Quad-core ARM Cortex-A53 up to 1.5GHz, NEON coprocessor	Quad-core ARM Cortex-A53 up to 1.5GHz
Real-Time Processing Unit	-	-	Dual-core ARM Cortex-R5 up to 600MHz
Multimedia Processing	GPU Pascal (256 cores, up to 1122 MHz)	-	GPU ARM Mali-400 (2 cores up to 667MHz)
Memory Interface	LPDDR4, 58.4 GBytes/sec	DDR4, DDR3, LP DDR3, 25.6 GBytes/sec	DDR4, LPDDR4, DDR3, DDR3L, LPDDR3
High-Speed Peripherals	PCIe 2.0, 2xDSI, eDP 1.4, HDMI 2.0, CAN, UART, SPI, I2C, I2S, USB 3.0, GbE,…	PCIe 3.0, 2xUSB 2.0, 3xGbE, 2xUART, 4xSPI , 5xI2C, 1x eMMC 4.5,...	PCIe 2.1, 2xUSB3.0, sATA 3.1, 4xGbE, 2x eMMC 4.51, 2xQSPI, 2xI2C, 2xCAN 2.0,…
Programmable Logic			
Max Logic Cells / System Logic Cells (K)	-	378	Up to 1,143
Technology	16 nm	14 nm	16 nm
Power	15 W	12,5 W	2- 24 W

Figure 12: Detailed comparison of the Tegra X2, the Stratix 10 and the Zynq UltraScale+ platforms.

3.3.4 Vendor Selection and Post Installation Review

Considering the constraints of the use cases and the trade-offs that have to be made, the Zynq Ultrascale+ MPSoC was chosen as the TULIPP hardware platform. It is the most versatile device, brings the best energy efficiency, allows to implement some of the functions of the application as hardware accelerators, and helps saving power by switching off the unused parts of the SoC. The implementations of the TULIPP use cases confirm that it was possible to meet requirements using the MPSoC platform.

Since the use cases have different I/O interface requirements, we chose to develop a carrier board for the TULIPP platform. An additional benefit of this choice is that we can move to more powerful computing modules when these become available. Technical solutions and steps needed for developing of such boards are discussed in the next section.

3.4 Processor Module Alternatives

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 Programmable Logic (PL) sizes.

While this gives the user many options for choosing the most appropriate 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 standardised interfaces. This allows the board manufacturer to develop carrier boards with a 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 with the global system when we follow the evolution of the chips.

There are several approaches but there is not yet a clear emerging standard. In this section, we discuss the most important approaches to assemble such a module and connect it to the main board.

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.

System on Module (SoM): A System on Module (SoM) 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 types 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. There are several ways to connect SoM to carrier boards (like physical connector and protocols), and it is difficult to select a single standard that would fit them all. Because SoMs allow for more control of the system design process and better stability, they also reduce design costs.

Embedded System Module (ESM): An Embedded System Module (ESM) typically includes a CPU, memory, module-specific I/O interfaces and a number of basic 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 x 71 mm (5.9 x 2.8 in).

PC/104: PC/104 (or PC104) is a family of embedded computer standards which define both form factors and computer busses [32]. PC/104 is intended for specialised 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 customised embedded system as shown in Figure 13. It currently relies on PCI and more recently (2008) on PCI-Express busses.

The TULIPP carrier board follows the PC104 standard. We chose this standard to simplify mounting the platform environments such as cars and UAVs. Another benefit of choosing the PC104 is that it supports stacking individual boards and thereby improve performance.

3.5 Input / Output Interface Alternatives

Depending on the application domain, the hardware has to interface with different kinds of sensors. For integration reasons, the hardware designer may be forced 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 domains there are fewer constraints, therefore more flexibility is allowed.

3.5.1 Camera Interfaces

The following interfaces are well suited for the TULIPP use cases:

- GigE Vision
- Camera Parallel Interface (CPI)
- CameraLink

GigE Vision: GigE Vision is an interface standard for high-performance industrial cameras [47]. It provides a framework for transmitting high-speed video and related control data over Ethernet networks. The distribution of software or development, manufacture or sale of hardware that implement the standard, require the payment of annual licensing fees. The

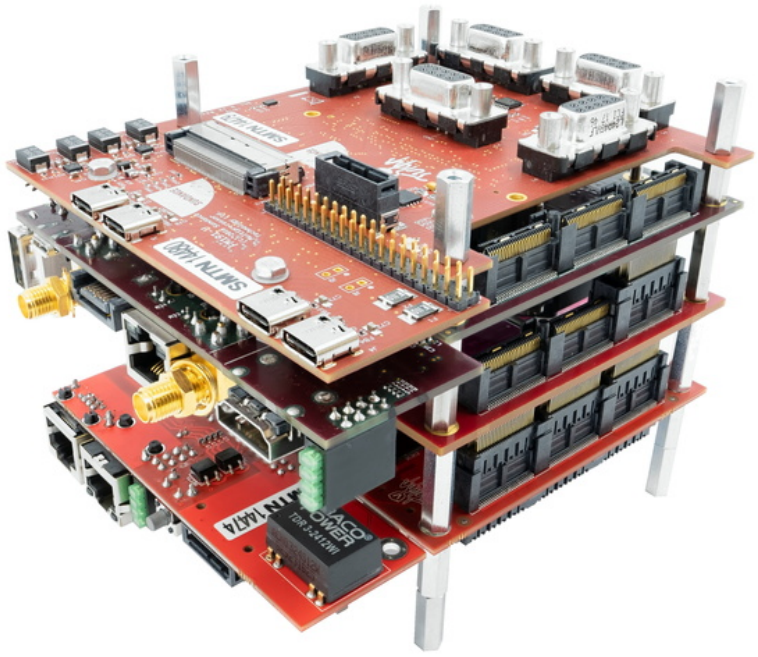


Figure 13: Connection of many PC/104 boards [32]

Automated Imaging Association (AIA) oversees the ongoing development and administration of the standard.

The GigE Vision standard consists of three parts from the user's point of view:

- Device Discovery makes it possible to find devices in the network.
- GigE Vision Control Protocol (GVCP) is used to camera control.
- GigE Vision Streaming Protocol (GVSP) provides a protocol for streaming both uncompressed and since version 2.0 compressed streams.

GigE Vision has the following technical characteristics:

- The standard is based on UDP/IP and makes multicast connections possible.
- It is mainly a transmission protocol standard and only describes communication to camera control and data streaming. Feature descriptions and access are controlled by the referenced GenICam standard.
- The GigE Vision standard does not restrict GigE Vision devices to cameras.

Camera Parallel Interface (CPI): CPI is one of the image sensor interfaces specified by the MIPI Alliance for various mobile product functions. It consists of two portions: an I2C

bus to control the interface and a parallel bus for the image data itself. The parallel bus is unidirectional.

CPI has the following technical characteristics:

- Uses 8 to 12 bits parallel data line to carry pixel data. The data transmitted on these lines change with every Pixel Clock (PCLK).
- Horizontal Sync (HSYNC) indicates that one line of the frame is transmitted, Vertical Sync (VSYNC) indicates that the entire frame is transferred.

CameraLink: CameraLink is an official standard managed by the Automated Imaging Association (AIA). The standard offers three different implementations: Base, Medium, and Full [49].

CameraLink has the following technical characteristics:

- The data transfer rate is in the range of 240 MB/s per chipset. For CameraLink Full, which uses three chipsets, this results in a maximum total throughput of about 674 MB/s.
- Cable length depends on camera speed: for lower speed cameras, cable lengths of up to 15 m are often possible. The cable length can be extended via repeaters and extenders.
- The standard has been extended by the function Power over CameraLink (PoCL) and CameraLink Lite.

3.5.2 Display Interfaces

The following display standards are well suited for TULIPP:

- High-Definition Multimedia Interface (HDMI)
- Display Serial Interface (DSI)

HDMI: HDMI is an interface standard for transferring of digital images and sound over cables. It supports image formats up to 8K @ 60 Hz and defines several different connector types. HDMI was initially developed by a group of companies and the standard is today maintained and developed by the organisation HDMI Forum. For manufacturers to implement HDMI in any product requires a license fee to be paid.

HDMI has the following technical characteristics:

- It defines both the protocols, signals, electrical interfaces and mechanical requirements.
- It has up to 5 different communication channels: DDC, TMDS, CEC, ARC, HEC.
  - The DDC channel is an i2c bus to exchanges information between the source and the sink device regarding image formats, etc.

  - TMDS is the actual data channel. It is packet based and interleaves video, audio and auxiliary data. Error detection and correction is embedded in the protocol – a required feature for medical devices.
  - CEC is a one bit serial bus enabling communication of control information between. devices, so several devices can be controlled from one remote control.
  - ARC is for routing of audio between devices.
  - HEC is an embedded 100 Mbit/s Ethernet channel.
- It defines a large range of image resolutions, clock rates, etc.
- it supports many different colour pixel configurations.

Display Serial Interface (DSI): DSI is a standard for driving display panels, normally at short range, typically within products like mobile phones, TVs, etc. It consists of a serial bus with one or more lanes and a separate clock. The standard is maintained by the MIPI alliance.

DSI has the following technical characteristics:

- It is based on high speed differential signalling over one or more serial links.
- All links are point to point and communicates from the host to the display.
- The first link can be reversed and this feature is used when information is requested from the display.
- It defines protocols for both sending commands and for transferring of image data.
- It defines two modes of operations, one low speed, low power for commands and one high speed for transferring of actual image data.

3.5.3 Host Communication Standards

An embedded system is often controlled from a host computer. During development and testing, there is a need to read out logs and other relevant data, and there might also be a need to send commands to the embedded system during test runs. Depending on the application, there might also be a need for communication with a host computer during normal use. As an example, a UAV might record telemetry data during flight, which can be downloaded to a host after landing.

For these situations, there must be an interface for connecting the embedded system to the host computer. There are a number of suitable standards with different characteristics. The interface standard should be chosen based on the following considerations:

- Performance: The performance spectrum of the different communication interfaces is huge. Identifying performance requirements are, therefore, important.
- Availability on host computer. It is often important that the embedded device can be connected to an ordinary PC without extra equipment. If this is the case, one of the interfaces found on common PCs should be chosen.

- Ease of implementation. Software stack complexity can be a limiting factor for small embedded systems with limited memory and processing power. Physical requirements, such as connector footprint size and high-speed or RF considerations can also influence choice of interface.
- Power capabilities; if the device should be powered from the communication cable
- Communication distance; how long is the distance to the host computer.

This section will go through the most common interface standards used for the purposes mentioned above, and discuss their characteristics. The are two main categories: Wired and wireless. The section will discuss both.

USB: USB is one of the most widely used interfaces for wired devices today and is the prime candidate for connecting most types of devices to a host computer.

USB has the following technical characteristics:

- High performance interface. USB 3.0 SuperSpeed supports up to 10 Gb/s.
- Widely available, present on all modern PCs.
- Complex interface, although software stacks exist for small embedded devices. If high performance is required, the software and physical design becomes more challenging.
- The host can provide power to the device over USB.
- USB cables can be maximum 3–5 meters (depending on transfer speed)

Ethernet: Ethernet is the most important wired interface for connecting a PC to a computer network. This should also be considered as a host communication option for an embedded device, although the device will then be connected to a computer network, reachable by any device on that network. This is also the largest advantage of Ethernet, the device is no longer required to be connected to only one computer at a time, and no cables must be switched over to access the device from different computers.

Ethernet has the following technical characteristics:

- High performance interface. The highest performing Ethernet standard today is 400 Gb/s, although 10 or 100 Mb/s is more likely for an embedded system.
- Widely available, although the infrastructure requirements are slightly higher than for USB. An Ethernet switch might be necessary, and some laptops are sold without an Ethernet port.
- Complex software stack, although TCP/IP stacks exist for small embedded systems.
- It is usually not possible to power the device through the Ethernet cable.
- Communication distance is long (100m cable is possible).

EIA RS232: EIA RS232 used to be the most widely used wired communication interface, and has been available since 1960.

RD232 has the following technical characteristics:

- Low performance. Usually up to 115200 b/s.
- Used to be widely available, although modern laptops no longer ship with an RS232 port. RS232 to USB converters are available.
- Very simple software stack, can be implemented easily using almost no memory and processing power. Simple physical implementation – although typical RS232 connectors have a large footprint.
- It is not possible to power the device over an RS232 cable
- Max cable length is around 15 m

There are other similar standards with slightly different properties:

- EIA RS422: Differential signals to enhance noise resistance, cable length (up to 1500 m) and speed (up to 10 Mb/s).
- EIA RS485: Similar to 422, but half duplex (2 wires) instead of full duplex (4 wires)

Wireless LAN: Wireless LAN (802.11) is an option. From a software developer and user perspective, wireless LAN will be quite similar to Ethernet.

Wireless LAN has the following technical characteristics:

- High performance, up to several Gb/s, although the Mb/s range would be more likely.
- Widely available. A WLAN router is necessary
- Complex software stack, although TCP/IP stacks exist for small embedded systems.
- No cable, so not a power source for the device
- Communication distance is usually up to a few tens of meters, but depends on local conditions

Bluetooth: Bluetooth is a wireless standard for short distance communication and can be used between an embedded device and a host computer. A Bluetooth Low Energy standard exists targeting low power devices.

Bluetooth has the following technical characteristics:

- Low to medium performance, from around 0.27 Mb/s up to 2.1 Mb/s depending on standard.
- Widely available; most laptops and some desktop computers come with Bluetooth, and USB adapters are available
- Complex software stack, although TCP/IP stacks exist for small embedded systems.
- No cable, so not a power source for the device
- Communication distance typically much less than 10 m

3.5.4 Peripheral Interfaces

Peripheral Component Interconnect (PCI) Express, officially abbreviated as PCIe or PCI-e, is a high-speed serial bus standard. PCI Express is based on point-to-point topology with separate serial links connecting every device to the root complex (host). It can consist of one to 32 lanes (1, 2, 4, 8, 12, 16, 32) and supports full-duplex communication between any two endpoints. A lane is composed of two differential signalling pairs (four wires or signal traces): one pair for receiving data and the other for transmitting. PCI Express communication is encapsulated in packets which are transported in byte format simultaneously in both directions between endpoints of a link. PCI Express 3.0 provides 8 GT/s bit rate, PCI Express 4.0 - 16 GT/s bit rate (8 GB/s total for x4 lanes)

3.5.5 Debug Interfaces

A debug interface is necessary during development. The debug interface gives access to the device while it is running, supporting all the typical debugger activities such as specifying breakpoints and controlling execution (stop, continue, single-step, etc.). In addition, the debug interface should support uploading and flashing binaries.

The selection of debugging interfaces is small and the choice is highly dependent on what the chosen target processor supports. Usually, the debug interface is not an important consideration (as long as it is available), and the embedded developer will accept whatever is provided by the chosen microcontroller.

JTAG: JTAG is the most widely used test and debug interface for all kinds of chips and systems. Although originally designed for board level testing of daisy chained integrated circuits, one of the main uses of JTAG today is for debugging software, providing all the capabilities mentioned above. JTAG is a 5 wire interface and supports daisy chaining devices such that several devices on a board can be debugged through the same JTAG port. A 2 wire variant exists, but it is not commonly used.

Serial Wire Debug (SWD): Not all microcontrollers support the JTAG interface, mainly to reduce the pincount needed for debugging. The main alternative is the ARM SWD interface which only requires two wires (with an optional third wire for console output). All the software debugging functionality normally provided over JTAG will usually also be available over SWD.

4 Operating System and Libraries

The TULIPP Reference Platform (TRP) provides an assemblage of hardware and software components. In this chapter, we justify the need in the TRP for an operating system and associated middleware libraries. As the previous chapters introduced the challenges of designing embedded systems and the hardware components, this chapter briefly introduces the run-time software components of the embedded platform. In Section 4.1, we present operating system concepts required in the context of image processing systems, and we describe what choices we made in the context of the TRP. In Section 4.2, we present middleware libraries selected in the context especially for their adequacy to image processing embedded systems.

4.1 Operating System

An embedded image processing system consists of a hardware platform (target platform, IP cores, etc.) and software (application code, operating systems, etc.). An important choice in the early design phase of the system is the Operating System (OS). The OS is a software component which is very close to the hardware, manages resources safely and efficiently, and provides services and a higher level of abstraction to application code. Broadly, there are three main strategies for OS selection:

- **Bare-metal systems:** These systems do not use an OS, and the application software runs directly on the processor in a privileged mode ("on the metal" as the name suggests). This means that scheduling, memory management, interrupt service routines, etc. are managed directly by the application. This paradigm allows for designing efficient systems with low overheads in terms of execution time and memory usage. However, the development costs are very high as every software layer of the system has to be designed and validated for the target product.
- **General Purpose Operating System (GPOS):** To avoid having to design and validate everything from scratch for every product, the application designers can decide to use an OS to manage the aforementioned low-level system features. For this strategy to be viable, the OS must support the chosen hardware platform. Then, application designers have to write their software such that it complies with and uses the APIs and paradigms of the chosen OS. In practice, this means using the system calls defined by this specific OS or using libraries that are ported on the OS. Additionally, using an operating system allows for a more structured approach relying on standard APIs which allows for faster development but also allows to build on the experience of previous users and developers. A key challenge is that GPOSes are based on "best-effort" strategies for the implementation of the different algorithms required – and cannot provide real-time guarantees. The lack of real time guarantees can lead to problems such as missed deadlines, priority inversion, deadlocks or starvation. These problems are undesirable in any system – but can result in loss of life in systems with hard real time requirements. In the case of image processing systems, it means that no strict guarantee on the quality of service can be provided.

- **Real-Time Operating System (RTOS):** An alternative to using a GPOS is to use an RTOS which is designed to guarantee the fulfilment of the real-time and reliability requirements of safety-critical applications. Timing requirements such as deadlines and execution budget can be encoded into the application and the RTOS scheduler takes these constraints into account at runtime.

In TULIPP, we chose to go for the RTOS option as all use cases have real-time requirements, which is also the case for most low-power image processing applications. HIPPEROS, which is the OS of the target platform instance, is a RTOS. It implements all the features necessary to ensure safe and efficient allocation of the platform resources to the application.

4.1.1 OS Requirements of Image Processing Systems

The application paradigm is based on image processing and machine vision. The target platforms chosen (based on Zynq-7000 and Zynq UltraScale+ architecture, as justified in Section 3.3.3) are embedded and very heterogeneous in nature. The objectives are to produce high performance yet low-power systems.

These systems, in the context of the TRP, will be built with run-time middleware frameworks and tools that facilitate their development and the matching of their requirements in terms of performance, energy efficiency, reliability and costs. These frameworks use modern hardware platforms with multiple cores that could be homogeneous or heterogeneous.

As stated in Paolillo's thesis [31], to ensure that these applications take advantage of the facilities offered by modern hardware platforms, the application and middleware frameworks require software support at the operating system level. More specifically, the underlying real-time kernel must support both multi-core platforms and power management. The following describes a non-exhaustive list of the features required by such a real-time operating system:

- **Multi-thread support:** The RTOS must be able to allocate several threads to a real-time task and controlling the priority of these threads with regards to the other task of the systems; in other words, the kernel scheduler must be able to manage threads as the basic units of scheduling.
- **Power management support:** Drivers and governors for Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM) must be available at kernel level, to take decisions accordingly with the thread scheduler and allow the system designer to implement a power management strategy.
- **Real-time scheduling support:** The kernel must be able to schedule threads according to a multi-core real-time scheduling policy. The TULIPP use cases have real-time constraints that need to be to be taken into account by the OS to ensure that the application reacts to external stimuli in a timely manner. A dedicated interrupt management system for communication (i.e., the input frames and the output buffers) must be implemented to ensure such a real-time behaviour.
- **Real-time guarantees:** Every operation in the operating system must be predictable and optimised for the worst-case (in opposition to being optimised for the average case

as in a GPOS). In particular, system calls must have a predictable and documented execution time. Additionally, interrupt jitter must be minimised. In TULIPP, the medical use case must not only react in real-time but must ensure that no frames are lost. 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 application deadlines and specific scheduling policies must be implemented in the operating system to describe those deadlines.

- **Parallelism support:** As any concurrent threads running in parallel could issue a system call at any time, the system must be able to concurrently enter the kernel mode on the different cores. Therefore, parallelism support requires designing a distributed architecture for the kernel execution. Examples of techniques include synchronisation methods and inter-core communication protocols.
- **Heterogeneous support:** Modern multi-core hardware platforms are more and more heterogeneous in nature. MPSoC platforms for embedded systems integrate several heterogeneous components, such as cores from different architectures and ISAs, reconfigurable hardware in the form of FPGAs, and various ASIC accelerators. To take advantage of all these heterogeneous components, operating system support is required. This support can take the form of device drivers and OS services in the simplest cases, but require major kernel components for the most complex cases. For example, in order to be able to run the kernel on cores of different target architecture, the kernel must be compiled for each architecture and ensure to have a compatible Application Binary Interface (ABI) and communication protocol across heterogeneous cores.
- **Low memory footprint:** The amount of memory is commonly a limiting factor in embedded low-power platforms. 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 hardware cost. Whatever the reason, the total memory is limited and the software must take this constraint into account. Since the OS is at the very heart of the software, it must leave enough space for the application binary and potential input data. 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 features that are required.
- **Library support:** The target OS must support libraries, APIs and device drivers related to the field of the systems that must be developed. In our case, image processing applications require extensive library support for parallel processing, image processing, camera input and output devices. The main selected libraries are discussed in Section 4.2.

This non-exhaustive set of features for an operating system for embedded applications (and its underlying kernel) would enable the design of parallel power-aware real-time applications (including image processing systems) for multi-core heterogeneous platforms. Table 2 presents a matrix of existing RTOSes and their respective features. None matches all the aforementioned requirements.

To summarise, the chosen RTOS must cope with these system characteristics and require-

Table 2: Matrix of features of existing RTOSes. None matches all requirements for high performance low-power embedded real-time applications.

	Requirements of market RTOSes							
Requirement	**FreeRTOS**	**PikeOS**	**RTEMS**	**VxWorks**	**QNX**	**Integrity**	**Nucleus RTOS**	**HIPPEROS**
Real-time scheduling	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Virtual memory support	No	Yes	No	Yes	Yes	Yes	Yes	Yes
Multi-core support	No	Yes	No	Yes	Yes	Yes	Yes	Yes
Multi-threading support	No	Yes	Yes	Yes	Yes	Yes	No	Yes
Heterogeneous processing	No	No	No	Yes	No	No	No	Yes
DVFS support	No	No	No	No	No	No	No	Yes
DPM support	No	No	No	No	No	No	Yes	Partial

ments. In practice, it means that the RTOS must support very modern hardware such as Multi-Processor System on Chip (MPSoC) platforms. Therefore, the RTOS must provides good support for parallelism: multi-threaded applications running on different cores, concurrency mechanisms (mutexes, semaphores, etc.), Additionally, the RTOS must implement low-power scheduling based on power management primitives such as DVFS and DPM.

4.1.2 Selecting the TRP OS

The chosen hardware platform (selected in Section 3.3.3) contains a reconfigurable FPGA substrate which can be used to instantiate application-specific accelerators. The RTOS must provide a service to make the FPGA substrate efficiently usable through providing a driver that enable configuring and communicating with the FPGA. These accelerators are very useful in the context of low-power and high performance image processing application: They allow implementing heavy image processing algorithms in hardware. Designing hardware accelerators allows to exploit inherent parallelism of hardware design and then design massively parallel solution to heavy computation problems. When processing an image, different regions of the image can be processed in parallel by the hardware accelerators, while in software the parallelism is limited by the number of cores and inherent instruction-level parallelism (ILP) present in the core architecture. However, FPGA-defined accelerators must be supported by the RTOS, together with support for the reconfiguration. Therefore, the chosen RTOS must provide run-time drivers for dynamic reconfiguration of the the FPGA Programmable Logic (PL) and device driver support for the application-specific accelerators.

To ensure system safety and reliability, the RTOS provides time and space isolation mechanisms, through a real-time scheduler and virtual memory management (with separation of virtual address spaces for application tasks).

Finally, the operating system must remain as small as possible so that it does not use all the available resources of the system both memory wise and computerise. Having an operating system fine-tuned for an application domain delivering the right and necessary libraries and APIs is the target reached by the TULIPP project.

In summary, the RTOS is a key component of the TULIPP platform instance. It allows to leverage the capabilities of the hardware to implement high performance yet low-power applications while respecting their safety-critical requirements.

In TULIPP, we selected the HIPPEROS RTOS as the best match for the first platform instance of the TRP. The main rationale for this is that HIPPEROS matches almost all the aforementioned criteria: It is a multi-tasking and multi-core micro-kernel-based real-time operating system for embedded platforms that provides support for FPGA and Dynamic Reconfiguration and that is adapted to image processing in the context of the TULIPP-HIPPEROS OS distribution.

More information about the HIPPEROS RTOS and how it has been defined for the TULIPP platform instance as a specific distribution are provided in deliverables D3.1, D3.2 and D3.3.

4.2 Libraries

As introduced in the previous section, low-power high performance image processing applications often require a large variety of libraries. In this section, we present a non-exhaustive list of libraries that we find important and consider the support to be high priority in the TRP.

4.2.1 Parallel Processing Libraries

In order to extract maximum performance and energy-efficiency from a MPSoC platform, parallel processing libraries must be used to leverage parallelism out of the platform at the application level. For embedded image processing systems, the two main alternatives are Posix Threads and OpenMP. Other parallel programming frameworks exist such as Cilk, MPI, Grand Central Dispatch and Threading Building Blocks [31], but we chose OpenMP and Posix Threads as priority libraries for parallel processing as they are the most important and promising regarding adoption, community support and ease of porting onto an embedded system.

Posix Threads: POSIX Threads, usually referred to as pthreads, is an execution model that exists independently from a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time [51]. Each flow of work is referred to as a thread, and creation and control over these flows is achieved by making calls to the POSIX Threads API. POSIX Threads is an API defined by the POSIX standard. Pthreads provide an API to program multi-threaded application on a POSIX-compliant OS.

OpenMP: OpenMP is standard framework that simplify the development of parallel multi-threaded applications. It consists of pragma statements that the developer adds to the application to control how it is parallelised. Based on these annotations, an OpenMP-enabled compiler and an OpenMP runtime cooperate to execute the application using multiple threads. Commonly, the runtime relies on the underlying OS for thread support, inter-processor communication mechanisms, and thread synchronization mechanisms. OpenMP

can be implemented using pthreads, and as a consequence offer a way simpler parallel application design framework than pthreads.

4.2.2 Image Processing Libraries

A number of image processing libraries have been proposed. The key idea is to improve developer productivity by providing high-performance implementations of image processing primitives. The decision of whether to use an image processing library depends strongly on the characteristics of the image processing application and the constraints of the product it will be deployed within (see Section 5.1 for a detailed discussion). That said, image processing libraries can be very useful when these constraints are met. In the remainder of this section, we discuss the OpenCV and OpenVX image processing libraries. In addition to these libraries, proprietary machine vision libraries and development environments exist (e.g., HALCON [27]).

OpenCV: Open Source Computer Vision Library provides building blocks for real-time image processing and computer vision applications. It includes basic functions such as stitching images together as well as state-of-the-art algorithms for human detection and tracking or producing a 3D point cloud from stereo vision systems. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. It is the reference library adopted in the image processing community and Xilinx recently released a vision library based on OpenCV. Its goal is to turn OpenCV calls into hardware accelerators (in the PL), making this library a target of choice for high performance low-power embedded image processing applications.

OpenVX: OpenVX is an open, royalty-free standard for cross platform acceleration of computer vision applications. It is a low-level programming framework, which efficiently enables access to computer-vision hardware accelerators for software developers. OpenVX aims to provide both functional and performance portability and enable performance and power-optimized computer vision processing. Its target are low power, real-time applications and portability across diverse heterogeneous processors. This is important for embedded and real-time use cases such as in smart video surveillance, object and scene reconstruction and advanced driver assistance systems (ADAS). Various vendors like AMD, NVIDIA, Cadence, and Synopsys support OpenVX and many other companies (e.g., Intel, Samsung, Texas Instruments, and Qualcomm) have contributed to the standard [15].

5 Tools for Productive Development of High Performance Embedded Image Processing Applications

The main objective of adding tool support for high performance embedded image processing development is to substantially reduce the development effort required to satisfy application requirements. The tools ensure that the developer can focus on core application development by automating recurring, but critical, tasks such as instrumenting code to gather performance profiles, design space exploration and vendor tool configuration. Thus, we are adopting a broad definition of the word tool-chain by including any tool that improves the efficiency of image processing application development. 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 frame arrival until processing is complete.

The development of image processing solutions generally follows the *Generic Development Process (GDP)* which was introduced in Section 1.2. GDP is an iterative process that generalises the approach taken by programmers when implementing highly efficient image processing applications. It starts with a functionally correct implementation of an image processing system which runs on a general-purpose processor. High-level partitioning decides 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. The purpose of the tool-chain is to minimise the number of iterations required and the time spent in each iteration before arriving at an implementation that meets requirements.

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 specialised by using accelerators [7]. This results in hardware platforms for image processing being heterogeneous by containing a collection of compute units with different performance and power consumption characteristics.

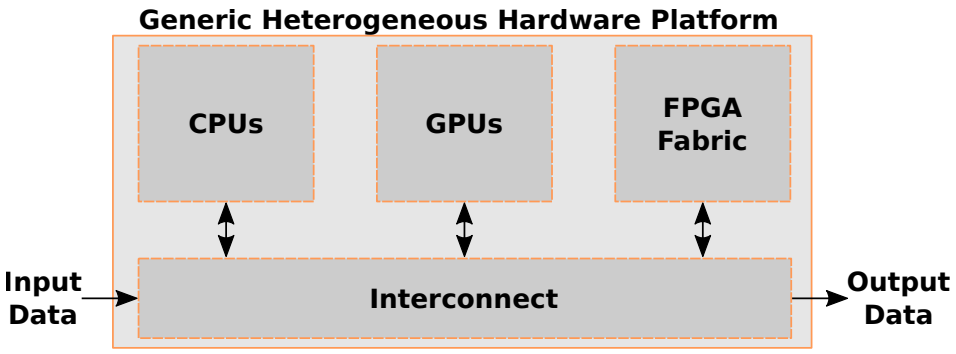


Figure 14: A Generic Heterogeneous Hardware Platform (GHHP)

Figure 14 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. In addition, the GHHP typically contains local and global memory structures that may or may not be visible to the developer. When discussing performance analysis tools in general, we envision that the target is a GHHP. When focusing on a particular project, we use the concept of a platform instance [20]. The platform instance is the hardware platform, OS and tools in use in the given project.

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 runtime/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 with profiling and automatically relate these to source code constructs. This capability also needs to cover both runtime/latency and energy/power.

Tools aiming at improving development productivity are not the only tools necessary for supporting the full project life-cycle. In addition, tools are necessary to 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 development productivity tools in this chapter.

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 productivity-enhancing tools – and in particular performance analysis tools – to enable quickly developing systems that meet application requirements. In this chapter, we introduce the state-of-the-art programming models for heterogeneous systems in Section 5.1, and discuss the key features needed for performance tools to efficiently support GDP in the context of embedded image processing systems development in Section 5.2. Then, we present the performance- and productivity-enhancing utilities developed in the TULIPP project in Section 5.3.

Table 3: Advantages and challenges of the SPM and MPM strategies.

SPM-strategy	Advantages	• Setup is simpler since a single tool chain can be used for the complete application. • 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.
	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.
MPM-strategy	Advantages	• Using specialised vendor tools for each component reduces the risk of introducing performance-limiting abstractions. • Platform selection is simplified since vendors can support different programming models.
	Challenges	• Application maintenance is complicated by multiple tool-chains, especially due to upgrades. • Development 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.

5.1 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 is 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.

Table 3 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.

5.1.1 Single Programming Model Strategies

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

- **SIMD-based models:** In these models, the part of the application that will be offloaded to the accelerator is rewritten as a kernel that performs the desired processing on a subset of the application's input data – a *Single-Instruction Multiple-Data (SIMD)* approach. In this way, the runtime can invoke a large number of threads – some of which are executed in lock-step – to exploit parallelism at a comparatively low hardware cost. Each thread is assigned a (possibly multi-dimensional) identifier which the kernel uses to select its input data. The main example of this model is OpenCL [42] which supports a range of devices – including GPUs and FPGAs. For GHHPs that include only GPUs and CPUs, NVIDIA CUDA [9] is another example.
- **High Level Synthesis (HLS):** The input to an HLS tool is an implementation of an application in a high-level programming language (commonly C or C++). Then, the programmer uses specific constructs to instruct the HLS tool to generate custom hardware implementations of performance-critical functions. HLS approaches are SPM-strategies for GHHPs that consist of CPUs and an FPGA fabric, and examples of HLS approaches [28] are Xilinx Vivado HLS [54], LegUP [8], and Bambu [33].
- **Library-based image processing:** In this approach, the programmer uses standard libraries such as OpenCV [22] or OpenVX [15] to implement performance critical image processing kernels. In this approach, the programmer does not need to deal with the characteristics of the of the computational devices on the GHHP and simply leverages optimised implementations provided by the libraries. However, the programmer is limited to the functionality supported by the libraries, and a high-performance implementation of the preferred library must be available on the chosen platform.

The SIMD-based models are desirable because they leverage a familiar parallel programming abstraction – i.e., Single Program Multiple Data (SPMD) – and are supported by a rich ecosystem of tools. OpenCL [42] is a standard API that enables program execution on GHHP containing 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 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 optimisation 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 [48].

The abstractions of OpenCL may limit implementation flexibility on hardware platforms that contain reconfigurable fabrics such as FPGAs. An alternative approach is 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 [54] and LegUP [8]). 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.

The library-based image processing strategy abstracts away how an image processing function is implemented, and we introduced the most important libraries for embedded image processing in Section 4.2. A library-based strategy is a great option when the functions of the library maps well to the performance critical regions of the application. If it does not, it is unlikely that a library-based approach will yield competitive performance. A common situation is that the invocation of separate kernels lead to excessive copying of data – resulting in excessive overhead that outweighs the performance improvement gained by accelerating the function.

5.1.2 Multiple Programming Model Strategies

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 which means that the programmer may need to respond to the architectural challenges that can arise on multi-cores [18, 19]. Programming models for multi-cores has been studied extensively, and powerful tools such as OpenMP [11] are available to efficiently parallelise 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 [24, 26].
- **GPU:** GPUs are commonly included in image processing platforms due to their high efficiency on graphics and image processing tasks, and GPU vendors tend to support using their hardware for general purpose computations. However, each vendor tends to promote a specific programming model. For instance, NVIDIA focuses on CUDA while AMD and ARM focus on OpenCL.
- **FPGA:** Development for FPGAs have traditionally used Register Transfer Level (RTL) programming 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 [2]. 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.

5.2 Selecting and 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. It is not useful to quickly develop 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 visualisations such as Gantt charts or Grain Graphs [26]. With appropriate mechanisms, the visualisations 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 [13].

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., [23, 34]). 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., [55]). 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.

5.3 STHEM: The TULIPP Performance and Productivity-Enhancing Utilities

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 and productivity-enhancing utilities that have been developed during the TULIPP project. Overall, we follow an HLS-based SPM-strategy and use a combination of state-of-the-art industrial tools and novel research-based utilities. However, we also support a library-based SPM-strategy since this strategy is very efficient when the functionality of the library is a good fit for the needs of the application.

The TULIPP toolchain is called STHEM – an acronym for *Supporting uTilities for Heterogeneous EMbedded image processing platforms* [13, 20, 38, 39]. At a high-level, STHEM

contains two types of tool packages:

- *Vendor Tools (VTs):* VTs are existing, industry-grade tool packages that are critical to enable GDP on a particular platform instance, and they are commonly supplied by platform vendors or third-party companies. For the TULIPP platforms, examples of VTs are the Xilinx SDSoC development environment and the HIPPEROS real-time operating system. VTs are commonly large and complex, and it is both infeasible and inefficient to not use them when they are available.
- *Utilities:* Utilities are smaller tool packages provided by the TULIPP project that are designed to resolve limitations that hamper developer productivity on a particular platform. To facilitate reuse across platforms, the utilities are designed to be as independent of the VTs as possible and are often stand-alone. A utility may consist of a single hardware or software tool, or a collection of several tools working together to provide a certain functionality.

The objective of STHEM is to provide efficient support for GDP on the TULIPP platform. To reach this objective, we leverage VTs when they are available to ensure that the baseline tool-chain is comparable to current state-of-the-art tool-chains. Then, we identify cases where executing GDP is unnecessarily cumbersome and develop utilities to address these productivity issues.

The baseline TULIPP tool-chain consists of the Xilinx SDSoC development environment and the HIPPEROS real-time operating system. Thus, it already has efficient support for accelerating performance-critical procedures with SDSoC's HLS capability and ensuring reliable real-time performance with HIPPEROS.

However, it is generally challenging to manually use HLS to accelerate applications functions. First, it is difficult – and thereby time-consuming – to establish which functions to accelerate. Second, it is also difficult to develop an HLS-based implementation that both has sufficient performance, acceptable power consumption and does not need more computational resources than are available in the chosen FPGA [3]. Third, developing high-performance I/O-controllers for commonly used high resolution camera and display interfaces can be challenging.

The final version of STHEM contains utilities that help alleviate these challenges. Concretely, STHEM consists of the following utilities:

- **The Power Measurement Utility (PMU)** gathers power consumption samples with high spatial (i.e., high sample rate) and temporal resolution (up to seven concurrent inputs) and relates them to application behaviour through non-intrusively sampling the program counter of the CPUs.
- **The Analysis Utility (AU)** enables visual and automatic analysis of application performance and power consumption based on the profiles collected by the PMU.
- **The Dynamic Partial Reconfiguration Utility (DPRU)** enables using dynamic partial reconfiguration within SDSoC.
- **The High-Level Synthesis FPGA Library for Image Processing using OpenVX (HiFlipVX)** includes highly optimized image processing functions that developers can

Table 4: The STHEM utilities and their key benefits.

Utility	Key Benefits	Status
PMU	• Reduces the time spent establishing system power and energy consumption by providing power profiles with high temporal and high spatial resolution. • Automatically correlates power samples with program counter values non-intrusively retrieved from the platform CPUs.	**Complete**
AU	• Reduces the time it takes the developer to identify performance and power consumption issues by visualising the performance and power profiles collected by the PMU. • Reduces development time with automatic *Design Space Exploration (DSE)* of HLS configurations – in contrast to time-consuming manual exploration of configuration options.	**Complete**
DPRU	• Reduces development time by enabling using HLS in systems that require to dynamically reconfigure the FPGA. Concretely, it enables using dynamic partial reconfiguration with SDSoC.	**Complete**
HiFlipVX	• Reduces development time by adding optimized FPGA-enabled implementations of commonly used image processing functions.	**Complete**
IIU	• Reduces development time by including support for cameras that support the CameraLink interface. • Reduces development time by readily supporting HDMI input and output.	**Complete**
FDU	• Provides lossless stream of signal values to facilitate on-FPGA accelerator debugging.	**Complete**

use to transparently accelerate their image processing application.

- **The I/O IP Utility (IIU)** includes Intellectual Property (IP) cores for camera input over the Camera Link interface as well as IPs for HDMI input and output.
- **The FPGA Debug Utility (FDU)** provides a lossless stream of signal data to simplify online FPGA debugging.

Collectively, the STHEM utilities improve support for GDP on the TULIPP platforms by streamlining the process of developing an image processing system that meets performance and power constraints. Table 4 summarises the key benefits of each utility. For further details regarding the utilities, we refer the reader to D4.4 [12].

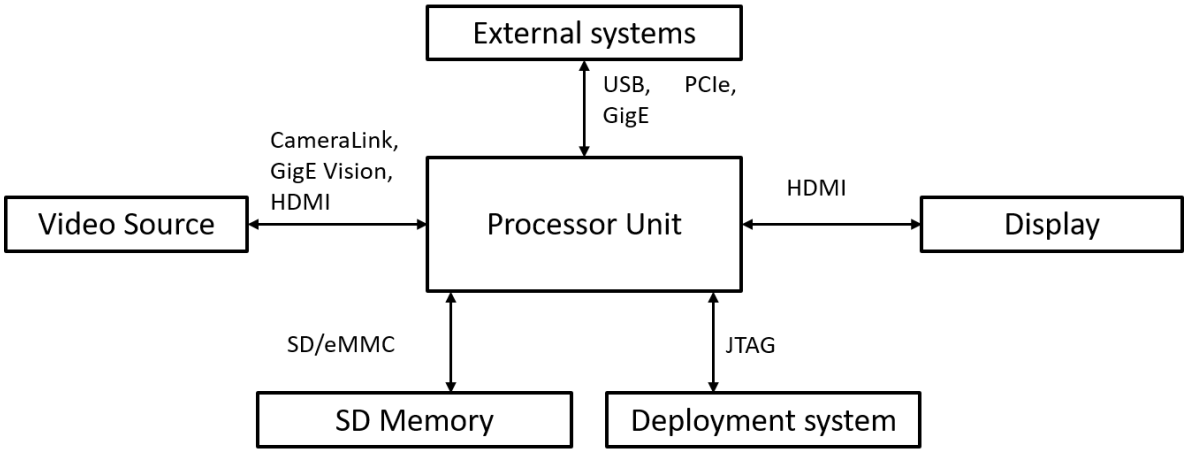


Figure 15: The hardware interfaces of the TULIPP reference platform.

6 Platform, Standards and Standardisation

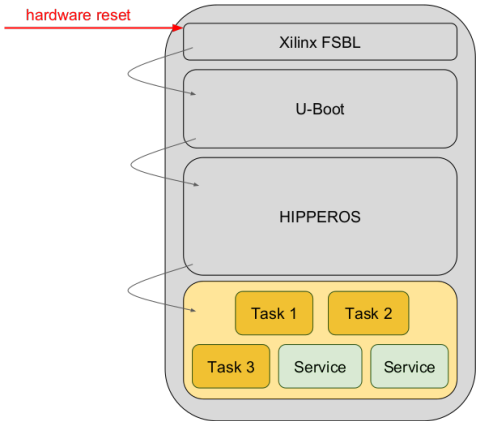
Standardisation is a critical means towards improving the productivity and competitiveness of the European high-performance embedded image processing industry. A key objective of TULIPP is to define a reference platform that future standardisation efforts can build upon. In this chapter, we describe the hardware, software and tool-chain interfaces we have selected for the TULIPP Reference Platform (TRP) (see Section 6.1, 6.2, and 6.3, respectively). The interfaces and approaches that were candidates to be included in the TRP were discussed in Chapter 3, 4, and 5.

6.1 The Hardware Platform Interfaces

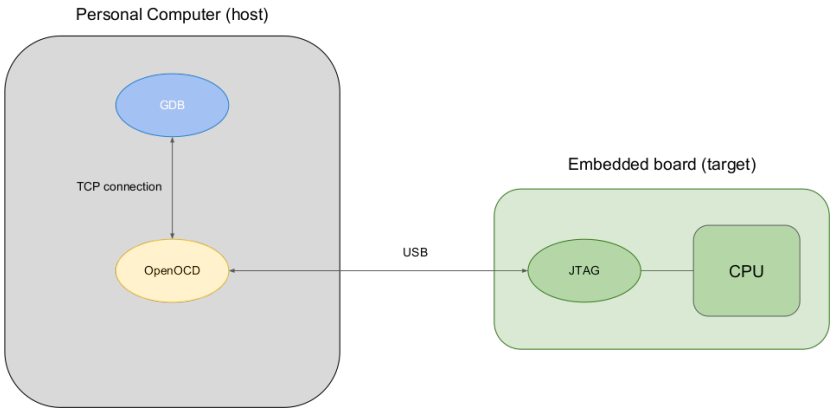
MPSoC Zynq UltraScale+ is the hardware platform selected for the TULIPP project (see Section 3.3). It implements many different interfaces (see Figure 11), some of which can be used both externally and internally. Figure 15 shows the interfaces that we support within the TULIPP Reference Platform (TRP) to connect to external components and systems. Each interface includes a hardware part and may also require software support (e.g., an OS driver).

Real-time image processing systems typically require a video input source. Normally, this is one or more cameras that implement generic interfaces such as GigE, USB or Firewire or specialised camera interfaces like Camera Parallel Interface (CPI), HDMI, CameraLink or GigE Vision. A display may be used to visualise the results.

It is also generally useful to have support for persistent storage. To this end, we support the SD/eMMC interface. For deployment and debugging, JTAG is used. In many use cases there are external systems which must be controlled based on image processing results – e.g, the UAV autopilot. Such systems can use different standard interfaces like USB or GigE. The other type of external systems are image processing boards, which may be needed to improve processing speed. Such boards can be connected via PCIe.



(a) Boot interfaces.



(b) Deploy and debug interfaces.

Figure 16: The run-time interfaces of the TRP.

6.2 The Operating System Interfaces

Because the application domain is high performance embedded image processing, we chose to add support for some of the most common libraries alongside the Operating System (OS). OpenCV has become a defacto standard for computer vision application and offers a set of libraries to efficiently implement such applications. While OpenCV is commonly used, it is not possible to compile the library and deploy it on an embedded architecture because its memory requirements exceed memory capacity of most hardware platforms. Therefore, we chose to support only a subset of OpenCV – focusing on the most useful functions for the TULIPP use cases. In addition, we provide a FPGA-accelerated versions of selected OpenVX functions.

To create a coherent and usable ecosystem, the OS must respect and/or implement a set of interfaces (or APIs). For the case of the TULIPP reference OS, we define the following interfaces (see Figure 16):

- **Development interfaces:** To develop tasks for the selected operating system distribution, application designers must write code in the *C or C++ languages*. The selected

OS should be bundled with an SDK including *toolchains and libraries* tailored to use the system calls TRP OS. Such libraries provided are implementations of the *Standard C Library, the Standard C++ Library and the Standard Template Library*. Toolchains are based on the *GNU GCC* compiler and *LLVM Clang*. Additionally, the OS provides a *driver library* to use the board devices (e.g. serial port, FPGA access to logic, etc.), a *file system implementation* to access the board on the SD card (based on the FAT32 standard file system), and a TCP/IP stack as a running service to provide *network access* to the applications. Additionally, libraries supporting parallelism and image processing are provided (OpenMP, OpenCV, OpenVX).

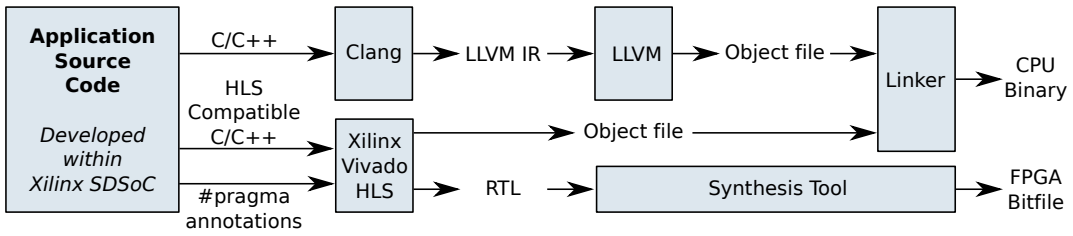
- **Deployment and debug interfaces:** To load software (and hardware for FPGA-enabled platforms) into the target platform, certain interfaces must be supported. The first interface that must be supported is the *bootloader*, and we chose to support mainly the U-Boot bootloader. U-Boot is an *Universal bootloader* for embedded systems, aiming at supporting a wide variety of embedded boards and target platforms. Choosing U-Boot for the TRP allows for better standard compliance as a wide range of boards on the market can therefore be supported. It is a *de facto* bootloader standard. The standard boot flow loads U-Boot, the HIPPEROS RTOS and the application from the SD card. Thus, the TRP needs file system support and we choose the FAT32 file system. We designed a user-friendly development flow: We developed a hot deployment mechanism based on the *JTAG interface* of the board. JTAG is a standard hardware interface to communicate with the remote target board. To this aim, on the host we use OpenOCD, which is a software that complies with JTAG and a wide range of boards and architectures. Additionally, the *serial port* (RS232, UART interfaces) is used for lightweight communication to drive the deployment process.

  To allow application designers to debug their code remotely on their target platform, we provide support for remote debugging. This is again based on the *JTAG* interfaces allowing to stop the processor while running, inspecting register values and memory locations, step into assembly or C code, etc. It is a key aspect of embedded software development.

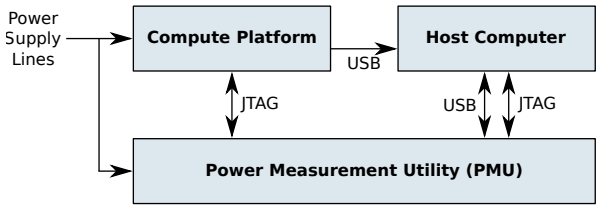
6.3 The Tool-Chain Interfaces

In the context of the TULIPP project, the tool-chain is the collection of tools that supports the *design time* activity of developing and optimising an embedded image processing application. Thus, they differ from the interfaces presented in Section 6.2 which were concerned with managing TRP resources at *runtime*. Broadly, the design time tools can be divided into three types:

- **Compilation tools** are responsible for transforming a program to a representation that can be executed on a computing platform. When the compilation tool generates a low-level hardware description, it is commonly called a synthesis tool. In addition, they tend to carry out extensive optimisation. These tools are a necessary component of a tool-chain because developing complex applications directly in a machine executable language is not feasible.



(a) Software interfaces.



(b) Hardware interfaces.

Figure 17: The design time interfaces of the TULIPP reference platform.

- **Verification tools** help the programmer verify that the implementation is correct. Examples for software are unit test frameworks while hardware implementations can be verified with simulators or formal verification tools. Verification tools are not strictly necessary since it is possible – but challenging – to develop applications without them.
- **Performance analysis tools** carry out advanced analysis of the application to identify performance, power or energy problems. These tools are also not necessary, but tracking down the root cause of performance issues without them can be time-consuming.

The TULIPP hardware platform contains an FPGA for application-specific acceleration (see Section 6.1). Acceleration is necessary to meet the performance and energy-efficiency requirements of embedded image processing tasks, and FPGAs have been shown to provide high performance at low power consumption for image classification tasks [45]. However, it is also well known that developing highly efficient FPGA-solutions is challenging [3]. In response to this situation, there has been a significant research effort towards developing *High Level Synthesis (HLS)* tools. HLS tools are able to transform an implementation in a high-level language (i.e., C or C++) into an accelerator circuit. Thus, HLS tools offer significantly improved productivity compared to traditional Register Transfer Level (RTL) design approaches which typically require developers so describe a plethora of low-level implementation details. Examples of state-of-the-art HLS tools are Xilinx Vivado HLS [54] and LegUP [8].

6.3.1 Compilation and Verification Tools

The aforementioned observations place a critical requirement on the TULIPP tool-chain – the need for including a state-of-the-art HLS tool. To satisfy this requirement, our tool-chain is based on Xilinx SDSoC (see Section 5.3), and Figure 17a shows an overview of the key interfaces within the tool-chain of the TULIPP Reference Platform (TRP). Xilinx SDSoC is a

complete development environment for compute platforms that tightly integrate CPUs with an FPGA-fabric. Thus, it provides direct access to both a state-of-the-art compiler for the CPU code and the Vivado HLS tool. This enables the the developer to simply select functions for acceleration on the FPGA-fabric. Then, the tool-chain will generate the required CPU binaries and FPGA bitfile required for running the application on the platform.

The default SDSoC tool-chain contains all necessary compilation and validation tools. Its C/C++ compiler and Xilinx Vivado HLS take a C/C++ implementation as input and emits executables. SDSoC requires that the developer uses command line arguments to select procedures for offloading to the FPGA with HLS. In addition, the developer can use pragma directives to control how each procedure is converted to RTL. Both the command line interface and pragma directives are Xilinx HLS specific. Standardising these interfaces would enable choosing different HLS tools within a single developer-facing tool.

6.3.2 Performance Analysis Tools

The objectives of the performance analysis tools are (1) to help the programmer choose which parts of the application to accelerate with the FPGA fabric and (2) to quantify the performance and power consumption of the application to determine if the current implementation satisfies requirements. Figure 17b shows the interfaces involved in gathering the performance and power profiles. The key component is the Power Measurement Utility (PMU) which we introduced in Section 5.3. The PMU samples power consumption by directly accessing the compute platform's power lines. In addition, it samples the Program Counter (PC) values of all CPUs over the JTAG interface (see Section 3.5.5). The PMU correlates the PC samples with the power consumption samples to non-intrusively produce highly detailed performance and power profiles.

To map the power and performance profiles to the program structure, we use Clang to generate an LLVM Intermediate Representation (IR) for the application (see Figure 17a). We use the LLVM IR to insert additional debug information into the CPU binary so that we later can identify which PCs occur within each basic block. This enables us to pinpoint the source code construct (e.g., function, for loop, etc.) that causes power and performance issues. Clang is the front-end for languages in the C-family in the LLVM [25] compiler infrastructure. Although the LLVM IR is not formally a standard, its specification is freely available.

In addition, JTAG is used to program the FPGA fabric. We leverage this capability to do automated Design Space Exploration (DSE) of the HLS configuration space. To enable this, the PMU supports passing commands issued by a Xilinx JTAG programmer through the PMU to the compute platform. However, JTAG can also be used to connect to the compute platform directly when the PMU is not used.

Finally, we use the Universal Serial Bus (USB) standard to communicate between the host computer and the PMU. The main use of this channel is to stream the power and PC samples to the host as they are collected. Within the TULIPP project, we have defined the format of this data stream. This format could be standardised to ensure inter-operability between different PMU-like devices. Finally, the compute platform uses USB to send terminal output to the host so that it can be shown to the developer.

7 Guidelines: Selecting Standards for Embedded Image Processing Systems

Ideally, embedded image processing applications need to process large amounts of data in real time while minimising the power consumption. Unfortunately, power consumption is fundamentally related to the hardware clock frequency [16]. This causes a challenging situation in which image processing applications need to be carefully tuned to satisfy both performance and power constraints.

The first step to achieve high power efficiency is to make sure that only functionality that is strictly necessary is included. This makes standardisation particularly challenging for embedded image processing systems since developers will not add features (such as standard compliance) that interfere with critical requirements (performance and power consumption). In TULIPP, we argue that the solution to this predicament is to define standards for key components (see Section 6) and then let the image processing application developer choose which of these standards to support. In this way, we can achieve the best of both: We reap the benefits of standardisation without incurring the overhead of supporting unnecessary functionality.

This approach requires a mechanism for selecting which interfaces to support in a particular embedded image processing system. In TULIPP, we use *guidelines* to aid the developer in making this choice. In this chapter, we first introduce the guideline concept (Section 7.1) before we discuss our methodologies for defining guidelines (Section 7.2) and improving guideline quality (Section 7.3). Appendix B contains a snapshot of the guidelines taken at the end of the TULIPP project.

7.1 The Guideline Concept

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.

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.

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 Section 7.2 and 7.3, respectively.

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

Table 5: Example of a guideline.

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 [1, 10, 35]
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 [30] to understand in more detail about exploiting vectorization and multithreading simultaneously.

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 heed to the encapsulated advice and considering the recommended implementation methods. Vendors ensure that the recommended methods are available in their products to attract developers. If implementing the recommended methods is infeasible due to constraints, then developers and vendors can consider alternative methods.

Guideline example: Table 5 contains an example of a guideline for obtaining high performance on multicore processors with vector units. Here, the advice orients 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.

Guideline compliance: 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 18. The output of instantiation process is an instance. The TULIPP Reference Platform (TRP) is an example of instantiation if the TULIPP consortium is considered to be a vendor.

A vendor-supplied 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 18 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

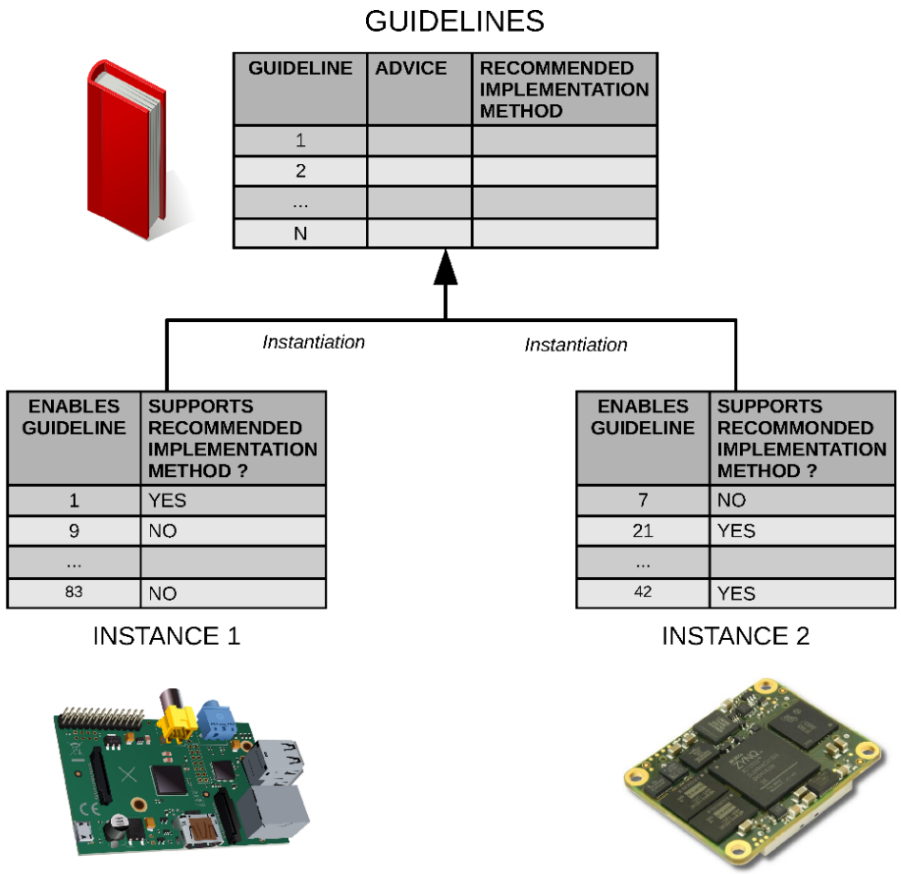


Figure 18: Instantiating platforms based on guidelines. The two instances are partially compliant with different subsets of guidelines.

is no longer relevant to the application and violates hard constraints such as cost and total power consumption.

7.2 Guideline Generation Methodology

A key contribution of the TULIPP project is a workflow to produce guidelines. 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. The workflow is illustrated in Figure 19. As evident from the illustration, the workflow captures a project-wide effort towards a common vision.

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

Image processing and embedded system 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 comes from experience and modelling. The expertise required to produce insights is not restricted to the project partners. External experts

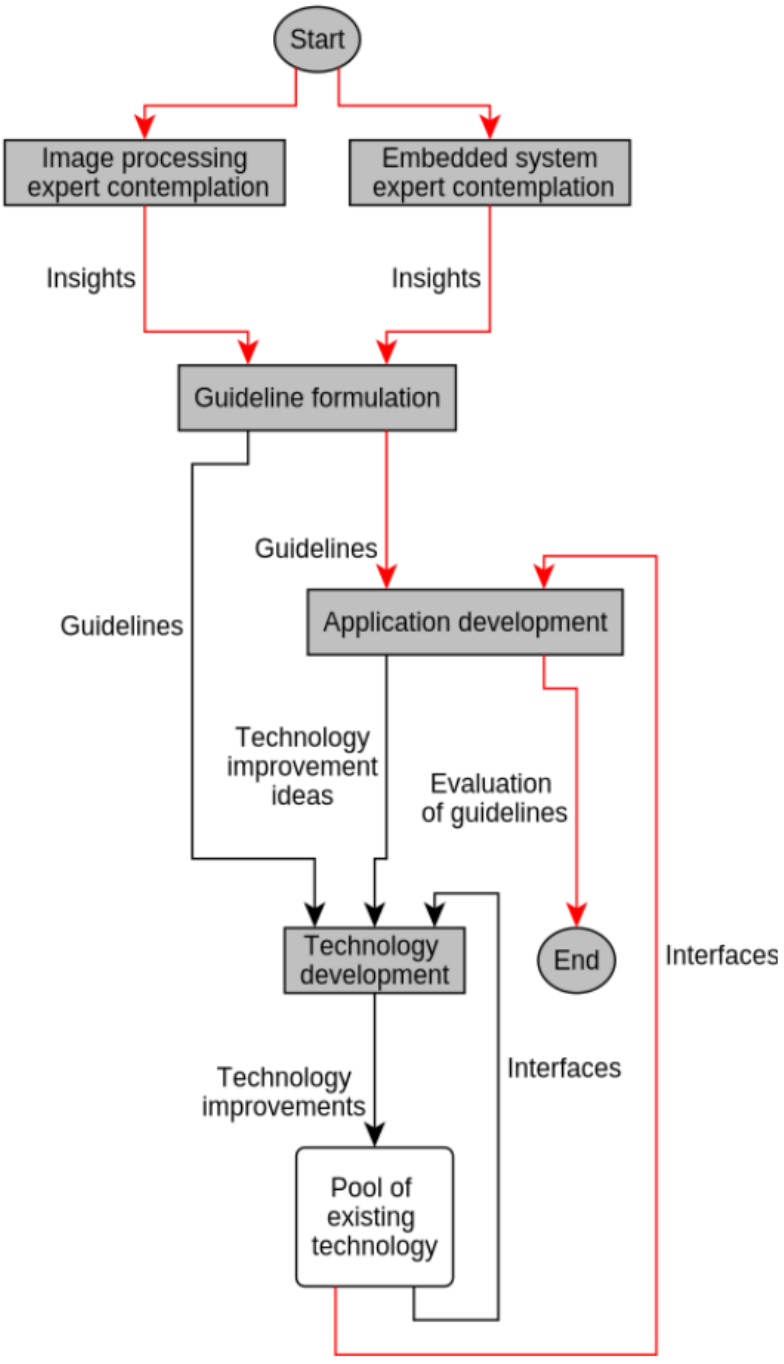


Figure 19: The workflow to generate and evaluate guidelines that define the platform instance. The critical path is shown in red.

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.

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

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

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.

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

> **Advice:** Balance between deterministic execution and performance while setting real-time OS (RTOS) parameters [40].
> **Recommended implementation method:** Configure the RTOS using vendor suggested methods [36]. Test and document candidate configurations extensively.

Application development: In this step, the formulated guidelines are implemented within the applications and evaluated for impact. 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. 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.

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 a level 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 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 realise that making practical use of the insights in a particular guideline requires technological support (e.g., though a tool or a driver).

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. To facilitate this, all guidelines are open-source and available on GitHub[1]. Appendix B contains a snapshot of the guidelines taken at the end of the TULIPP 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 and improve the general usefulness of the guidelines.

7.3 Guideline Quality Improvement Methodology

Generating new guidelines is not straightforward. The main difficulty is to define insightful guidelines that will impact a wide number of developers. While the guideline-creation process start from a particular practice or issues, a more general and global view of the problem as well as a higher level of information content is required for a guideline to be meaningful. Figure 20 shows our four-step methodology to derive generally useful guidelines from the individual insights had while developing platforms and applications. In the remainder of this section, we discuss each step in detail.

Step 1 – The Guideline Evaluation Board (GEB): To assess the quality of the collection of guidelines, the first step is to identify to whom the guidelines is intended – e.g., application

[1] https://github.com/tulipp-eu/tulipp-guidelines/wiki

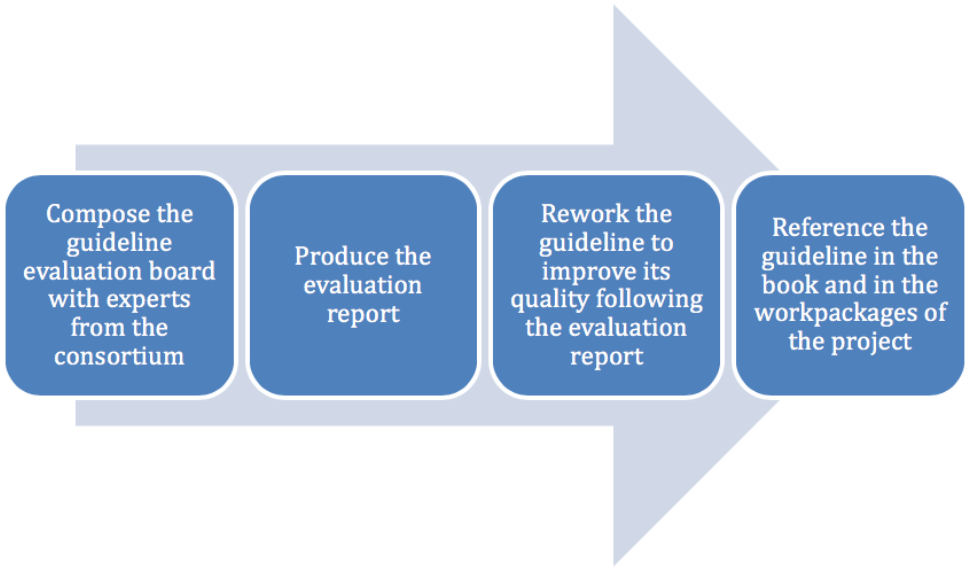


Figure 20: Guidelines quality improvement process.

developer or hardware designer. The reason is that a person within this group is best qualified to evaluate the insightfulness of the guideline. Note that more than one person might be need to evaluate a guideline to ensure that all audience groups are covered. We refer to this group as the Guideline Evaluation Board (GEB).

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(s) in the consortium. When the leader of WP1 cannot identify the type of persons addressed by the guideline, the executive board of TULIPP 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 evaluation board composed of experts from at least one of the following groups:

- **Hardware designers:** This group deals with the design of the hardware platform, the component selection, and component interfaces according to system requirements.
- **Operating System (OS) designers:** This groups deals with OS design and development. Examples are defining APIs for applications, ensuring that the OS works on a particular hardware platform, and providing the application with the means to efficiently control hardware behaviour.
- **Tool-chain designers:** This group deals with supporting the application developers by providing tools that automate recurring tasks. They must supply a comprehensive tool-chain that helps the application designer to efficiently map the application onto the hardware platform.
- **Application developers:** This group consists of experts in image processing. They know the algorithms used in an application and have the ability to implement this algo-

rithm on the chosen hardware platform. They understand the complete software stack, and can leverage tool support to faster develop the application.

- **System architect:** This group deals with the complete system definition. They are involved in a broad set of issues from the identification of the constraints that come from the final product, making sure that the system adheres to its price constraints, and are able to understand integration issues that arise due to choices made by the four other groups.

Step 2 – Evaluation report: The outcome of the GEB evaluation is an evaluation report that assesses the quality of the guidelines. The report has several objectives:

- It identifies the group(s) to which the guideline is intended.
- It identifies the domain of expertise required to write the guideline.
- It points out which parts of the guidelines are too restrictive or too use-case specific.
- It identifies the parts of the TULIPP handbook that are relevant to the given guideline.
- It identifies the work done in TULIPP that can benefit from the guideline.

The evaluation reports of each guideline are available in the guideline wiki[2].

Step 3 – The Guideline Improvement Expert Board (GIEB): Following the evaluation report and the identification of the required domains of expertise, the GEB assigns one or more experts of the topic addressed by the guideline to the Guideline Improvement Expert Board (GIEB). The GIEB is in charge of addressing the comments of the evaluation report and thereby improving the quality of the guideline. When the GIEB consists of more than one expert, one of the experts will be assigned to be the leader of the guideline improvement process. This person is responsible for integration of the contributions of others.

Step 4 – The guideline database and guideline references: When the guideline improvement process is completed, the guideline is integrated to the final guideline database. A snapshot of the guideline database taken at the end of the TULIPP project is provided in Appendix B.

[2] https://github.com/tulipp-eu/tulipp-guidelines/wiki

8 Conclusion

In this deliverable, we have presented the final version of the TULIPP Reference Platform (TRP). The TRP is defined by a collection of interfaces that interconnect a set of components that collectively provides productive development of high-performance, energy-efficient and real-time embedded image processing applications. Since image processing applications commonly push the performance limits of the embedded platforms, the TRP is augmented with a collection of guidelines. These guidelines enable developers to build on the insights of experts and reason about which interfaces and components are necessary for a particular image processing application – and which can be removed to reduce power consumption and design complexity. Both the TRP and the guidelines have been thoroughly evaluated on representative applications from the medical, automotive and UAV domains.

Overall, the TULIPP Reference Platform (TRP) fulfils objective 1 of the TULIPP project:

Objective 1: Define a reference platform for low-power image processing applications.

The TULIPP Reference Platform (TRP) combines highly power-efficient hardware with a state-of-the-art real-time operating system and a productivity-enhancing tool-chain. To build a basis for future standardisation efforts, we have clearly specified the interfaces between the key hardware and software components. In the remainder of this chapter, we first summarise the key interfaces of the TRP in Section 8.1. Then, we discuss how the TRP can be leveraged within future standardisation efforts in Section 8.2.

8.1 The TRP Interfaces

Table 6 lists the interfaces of the TULIPP Reference Platform (TRP). The key interfaces are concerned with image acquisition (e.g., CameraLink) since this is necessary for any image processing platform. In addition, image output (e.g., HDMI) is useful in many context – either for interfacing with the user or for demonstration or validation purposes. The hardware platform also commonly needs to support a number of communication interfaces – either with a host computer (e.g., USB, UART), between compute platforms (e.g., PCI), or connect the platform to the internet (e.g., Ethernet, WLAN). Finally, many tools require hardware support for debugging (e.g., JTAG).

The Operating System (OS) interfaces were introduced in Section 6.2. The OS makes it simpler for applications to use the hardware platform – by providing efficient resource management and ensuring safety and reliability. To integrate the OS in an embedded image processing ecosystem, a handful of interfaces must be respected, and for each interface the associated protocols and tools must be selected.

The tool-chain interfaces were introduced in Section 6.3. The tool-chain takes care of transforming the image processing application into executable that can be run on the hardware platform, as well as provide feedback to the developer on key performance indicators such as frame rate and power consumption. To enable this, a number of independent components need to interact. As mentioned in Chapter 5, we take a Single Programming Model (SPM) approach to application development which means that a single high-level language (i.e.,

Table 6: The interfaces of the TULIPP Reference Platform (TRP).

	TRP Interface	TRP Choice
Hardware Platform	Camera Interfaces	CPI, GigE Vision, CameraLink, HDMI
	Display interfaces	HDMI, DSI
	Communication interfaces	USB, PCIe, GigE
	Deployment interface	JTAG (IEEE 1149.1)
Operating System	Tasks implementation	C/C++
	File system	Fat32
	Network	TCP/IP, UDP/IP
	Bootloader interface	U-Boot
	Debug interface	JTAG (IEEE 1149.1)
	Parallel processing libraries	OpenMP, Posix threads
	Image processing libraries	OpenCV, OpenVX
Tool-chain	Programming language	C/C++
	Compilation and synthesis tools	GCC, LLVM, Xilinx Vivado
	Accelerator generator control	Vivado HLS annotations
	Software instrumentation interface	LLVM IR
	Object file linker	Xilinx linker
	PC sampling interface	JTAG (IEEE 1149.1)

C/C++) is the programming language interface. To improve power-efficiency and performance, critical parts of the application needs to be accelerated using the FPGA fabric. This is enabled through the accelerator generation tool and object file linker interfaces. Finally, the tool-chain enables identifying the source code construct (e.g., function, for loop, etc.) that causes power and performance issues. This is enabled through the software instrumentation and debug interfaces. The host-to-platform communication interface enables the developer monitor the progress and status of the application from the host.

8.2 The TRP and Standardisation

The TULIPP Reference Platform (TRP) could be proposed as a standard for low-power image processing applications in its own right – offering improved inter-operability between hardware and software components and enabling reuse across products. Unfortunately, it is likely that such a standard would not be widely adopted. The main reason is that embedded image processing applications commonly push the hardware platforms to their performance limits. Thus, there is no room for adding unnecessary overhead to the implementation. For this reason, a successful standard for low-power embedded image processing applications will need to be extremely lean.

A more viable strategy is to develop separate standards for each of the key interfaces of the TRP. Table 6 shows that many of the interfaces are already formally standardised or de-facto industry standards. However, the following interfaces of the TRP need to be further specified and standardised to improve interoperability:

- The interface between the application and High Level Synthesis (HLS) tool is highly vendor and tool dependent. Openly specifying this interface would enable seamlessly moving applications between different HLS tools.

- Tools that operate within the same ecosystem should be integrated to facilitate efficient system design. The interface between the OS and the Interactive Development Environment (IDE) should therefore be a standard. For IDEs enabling software acceleration, the design tools insert a run-time library in the flow between applications and the OS. This is often a closed library (e.g., `sds_lib` in the case of Xilinx SDSoC), but it should be open and standardised to ease integration of tools.

The existence of a collection of different – possibly overlapping and competing – standards creates the need for a way of selecting the standards to include in a particular embedded image processing application. To facilitate this process, we have proposed the concept of *guidelines*. A guideline encapsulates an expert insight in a precise, context-based formulation which orients the follower towards a goal by recommending an implementation method. Within TULIPP, we have defined a number of guidelines that we hope will help future developers to productively develop efficient embedded image processing applications (see Appendix B).

Bibliography

[1] ARM. NEON. `http://www.arm.com/products/processors/technologies/neon.php`, 2016.

[2] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: Constructing hardware in a scala embedded language. In *Proceedings of the Annual Design Automation Conference (DAC)*, pages 1216–1225, 2012.

[3] David F. Bacon, Rodric Rabbah, and Sunil Shukla. FPGA programming for the masses. *Commun. ACM*, 56(4):56–63, 2013.

[4] Z. K. Baker, M. B. Gokhale, and J. L. Tripp. Matched filter computation on fpga, cell and gpu. In *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*, pages 207–218, April 2007. doi: 10.1109/FCCM.2007.52.

[5] Michael Barr. Real men program in C. `https://webpages.uncc.edu/~jmconrad/ECGR4101Common/Articles/Real%20men%20program%20in%20C.pdf`, 2009.

[6] bertendsp. White paper: GPU vs FPGA Performance Comparison. `http://www.bertendsp.com/pdf/whitepaper/BWP001_GPU_vs_FPGA_Performance_Comparison_v1.0.pdf`, 2016.

[7] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67, 2011.

[8] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Tomasz Czajkowski, Stephen D. Brown, and Jason H. Anderson. 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, 2013.

[9] Shane Cook. *CUDA programming: A developer's guide to parallel computing with GPUs*. Newnes, 2012.

[10] Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz, and Robert W. Stewart. *The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Strathclyde Academic Media, 2014.

[11] L. Dagum and R. Menon. OpenMP: An industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998. ISSN 1070-9924.

[12] Asbjørn Djupdal, Björn Gottschall, Ariel Podlubne, Lester Kalms, Ahmed Sadek, Habib ul Hasan Khan, Muhammad Ali, Ahmed Kamal, Diana Goehringer, and Magnus Jahre. D4.4: Final tool chain. Technical report, TULIPP, 2018.

[13] Asbjørn Djupdal, Ananya Muddukrishna, and Magnus Jahre. D4.2: Tool chain with analysis module. Technical report, TULIPP, 2018.

[14] Jeremy Fowers, Greg Brown, Patrick Cooke, and Greg Stitt. A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA, pages 47–56, 2012.

[15] Radhakrishna Giduthuri and Kari Pulli. OpenVX: A framework for accelerating computer vision. In *SIGGRAPH ASIA 2016 Courses*, SA '16, pages 1–50, 2016.

[16] Mark Horowitz, Thomas Indermaur, and Ricardo Gonzalez. Low-power digital design. In *Symposium on Low Power Electronics*, pages 8–11, 1994.

[17] Intel. Intel Stratix 10 SoC FPGAs. `https://www.intel.com/content/www/us/en/products/programmable/soc/stratix-10.html`, 2018.

[18] M. Jahre, M. Grannaes, and L. Natvig. A quantitative study of memory system interference in chip multiprocessor architectures. In *11th IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2009.

[19] Magnus Jahre and Lasse Natvig. Performance effects of a cache miss handling architecture in a multi-core processor. In *Proceedings of the Norwegian Informatics Conference (NIK)*, 2007.

[20] Magnus Jahre, Asbjørn Djupdal, Lester Kalms, and Ananya Muddukrishna. D4.1: Basic tool chain. Technical report, TULIPP, 2017.

[21] J. P. Laudon K. C. Laudon. *Management Information Systems: Managing the Digital Firm.* Pearson Education, 2014.

[22] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: Computer vision in C++ with the OpenCV library*. "O'Reilly Media, Inc.", 2016.

[23] David Koeplinger, Christina Delimitrou, Raghu Prabhakar, Christos Kozyrakis, Yaqi Zhang, and Kunle Olukotun. Automatic Generation of Efficient Accelerators for Reconfigurable Hardware. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 115–127, 2016.

[24] Peder Voldnes Langdal, Magnus Jahre, and Ananya Muddukrishna. Extending OMPT to Support Grain Graphs. In *International Workshop on OpenMP (IWOMP)*, Lecture Notes in Computer Science, pages 141–155, 2017.

[25] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, 2004.

[26] Ananya Muddukrishna, Peter A. Jonsson, Artur Podobas, and Mats Brorsson. Grain Graphs: OpenMP Performance Analysis Made Easy. In *Proceedings of the Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 1–13, 2016.

[27] MVTec. Halcon. `https://www.mvtec.com/products/halcon/`, 2018.

[28] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels. A survey and evaluation of FPGA high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(10):1591–1604, 2016.

[29] NVIDIA. Get Under the Hood of Parker. `https://blogs.nvidia.com/blog/2016/08/22/parker-for-self-driving-cars/`, 2016.

[30] OpenMP. OpenMP specifications. `http://openmp.org/wp/openmp-specifications/`, 2016.

[31] Antonio Paolillo. *Optimisation of Performance Metrics of Embedded Hard Real-Time Systems using Software/Hardware Parallelism*. PhD thesis, Université libre de Bruxelles, Brussels, Belgium, 2018.

[32] PC/104. PC/104 Boards. `http://pc104.org`, 2018.

[33] C. Pilato and F. Ferrandi. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In *International Conference on Field programmable Logic and Applications (FPL)*, pages 1–4, 2013.

[34] Raghu Prabhakar, David Koeplinger, Kevin J. Brown, HyoukJoong Lee, Christopher De Sa, Christos Kozyrakis, and Kunle Olukotun. Generating Configurable Hardware from Parallel Patterns. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 651–665, 2016.

[35] Nikola Rajovic, Paul M. Carpenter, Isaac Gelado, Nikola Puzovic, Alex Ramirez, and Mateo Valero. Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC? In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2013.

[36] RTwiki. RT PREEMPT HOWTO. `https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO`, 2016.

[37] B. Ruf, S. Monka, M. Kollmann, and M. Grinberg. Real-time on-board obstacle avoidance for UAVs based on embedded stereo vision. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-1:363–370, 2018.

[38] Ahmad Sadek, Ananya Muddukrishna, Lester Kalms, Asbjørn Djupdal, Ariel Podlubne, Antonio Paolillo, Diana Goehringer, and Magnus Jahre. Supporting utilities for heterogeneous embedded image processing platforms (STHEM): An overview. In *Applied Reconfigurable Computing (ARC)*, 2018.

[39] Ahmed Sadek, Tobias Kalb, Diana Goehringer, and Magnus Jahre. D4.3: FPGA communication and reconfiguration IP. Technical report, TULIPP, 2018.

[40] Chris Simmonds. *Mastering Embedded Linux Programming*. Packt Publishing Ltd, 2015.

[41] Statista. Microvision. https://www.slideshare.net/MicroVision/mems-and-sensors-in-automotive-applications-on-the-road-to-autonomous-vehicles-hud-and-adas, 2016.

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

[43] Lars Struyf, Stijn De Beugher, Dong Hoon Van Uytsel, Frans Kanters, and Toon Goedemé. The battle of the giants – A case study of GPU vs FPGA optimisation for real-time image processing. In *PECCS*, 2014.

[44] O. Ulusel, C. Picardo, C. B. Harris, S. Reda, and R. I. Bahar. Hardware acceleration of feature detection and description algorithms on low-power embedded platforms. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–9, 2016.

[45] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, pages 65–74, 2017.

[46] M. Verhelst and B. Moons. Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to IoT and edge devices. *IEEE Solid-State Circuits Magazine*, 9(4):55–65, 2017.

[47] Visiononline. GiE Vision Standard. https://www.visiononline.org/vision-standards-details.cfm?type=5, 2018.

[48] Z. Wang, B. He, W. Zhang, and S. Jiang. A performance analysis framework for optimizing OpenCL applications on FPGAs. In *International Symposium on High Performance Computer Architecture (HPCA)*, pages 114–125, March 2016.

[49] Wikipedia. Camera Link. https://en.wikipedia.org/wiki/Camera_Link, 2018.

[50] Wikipedia. Weighted sum model. https://en.wikipedia.org/wiki/Weighted_sum_model, 2018.

[51] Wikipedia. Posix threads — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=POSIX_Threads&oldid=867704234, 2018.

[52] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009.

[53] Xilinx. Zynq Ultrascale+. https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html, 2018.

[54] Xilinx. Vivado High-Level Synthesis, 2018. URL `https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html`.

[55] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar. Design Space exploration of FPGA-based accelerators with multi-level parallelism. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1141–1146, 2017.

A Acronyms

API	Application Programming Interface
AU	Analysis Utility
CPI	Camera Parallel Interface
CPU	Central Processing Unit
DSE	Design Space Exploration
DSI	Display Serial Interface
DPM	Dynamic Power Management
DPRU	Dynamic Partial Reconfiguration Utility
DVFS	Dynamic Voltage and Frequency Scaling
ESM	Embedded System Module
FDU	FPGA Debug Utility
FPGA	Field Programmable Gate Array
GDP	Generic Development Process
GEB	Guideline Evaluation Board
GHHP	Generic Heterogeneous Hardware Platform
GIEB	Guideline Improvement Expert Board
GPOS	General Purpose Operating System
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface
HiFlipVX	High-Level Synthesis FPGA Library for Image Processing using OpenVX
HLS	High Level Synthesis
HPS	Hard Processor System
I/O	Input/Output
IDE	Interactive Development Environment
IIU	I/O IP Utility
IP	Intellectual Property
IR	Intermediate Representation
JTAG	Joint Test Action Group
MPM	Multiple Programming Model

MPSoC	Multi-Processor System on Chip
OS	Operating System
PC	Program Counter
PCI	Peripheral Component Interconnect
PL	Programmable Logic
PMU	Power Measurement Utility
PS	Processing System
ROP	Raster Operations
RTOS	Real-Time Operating System
RTL	Register Transfer Level
SIMD	Single-Instruction Multiple-Data
SoC	System on Chip
SoM	System on Module
SPM	Single Programming Model
SPMD	Single Program Multiple Data
STHEM	Supporting uTilities for Heterogeneous EMbedded image processing platforms
SWD	Serial Wire Debug
TRP	TULIPP Reference Platform
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
VT	Vendor Tool
WLAN	Wireless Local Area Network

B Guidelines

B.1 Adjust Your Algorithm to the Underlying Architecture

B.1.1 Guideline Information

Guideline Responsible	Boitumelo Ruf, Fraunhofer
Guideline Reviewer	Lester Kalms, TUD
Guideline Audience	Application developers, System architect
Guideline Expertise	Application Developers
Guideline Keywords	Code Optimisation, FPGA, GPU

B.1.2 Guideline advice

Adjust your 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. Optimize and tailor your algorithm to meet the strengths of the chosen architecture.

B.1.3 Insights that led to the guideline

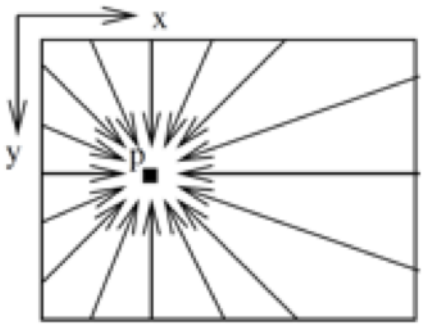


Figure 21: Illustration of the concentric optimization paths of the semi-global-matching algorithm. [1]

The semi-global-matching (SGM) aglorithm [1], which is used as part the stereo processing in the UAV use-case, is initially proposed to optimze the disparity value for a given pixel $\mathtt{p}$ in the disparity map along concentric paths over the whole image (cf. Figure 21). This strategy requires access to the whole image data for each pixel $\mathtt{p}$. While such an access pattern is no restriction for the use of general purpose computation on a GPU (GPGPU), it is, at least in its initial formulation, inappropriate for streamed processing on FPGAs.

However, multiple studies ([2]-[6]) have shown that the optimization strategy can be adjusted in order to facilitate the processing of the input data in a streamed manner.

B.1.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

The different architectures have different strengths. Their suitability greatly depends on the needs:

FPGA

1. FPGAs yield best performance-per-watt. However, the development of algorithms for FPGAs is cumbersome and require a high level of expert knowledge.
2. FPGAs have limited memory compared to a CPU and a GPU. Hence, not ideal for algorithms that require alot of memory!
3. Power consumption of FPGAs depends on used clock frequency. This allows meeting specific requirements without changing the algorithm, e.g. less power consumption by reducing the frame rate.

GPU

1. GPUs are more flexible than FPGAs in the choice of parameters during runtime and the kernel execusion.
2. General pupose computation on a GPU (GPGPU) is easily achieved by utilizing APIs such as OpenCL or CUDA.

B.1.5 Instantiation of the recommended implementation method in the reference platform

A literature survey on adaptations of the SGM algorithm to different architectures yields helpful insights for implementation of the algorithm for the UAV use-case. A List of appropriate literature can be found in References. Note that a reduction of the algorithms complexity, in order to adjust it to the underlining Hardware, might reduce its accuracy.

FPGA

In order to utilize the streaming characteristics of FPGAs, the aggregation paths should be restricted to the already processed pixels, i.e. 4 paths (upper left, upper, upper right, left and current pixel). While the use of eight paths require full image access for each pixel being aggre gated, the use of only four paths can efficiently be computed while streaming the pixel data and accumulating previously computed costs (cf. light blue pixels in Figure 22). This reduces complexity and accuracy of the algorithm, but increases computational speed significantly. Studies done in [2] show that a reduction from eight to four paths lead to a loss in accuracy of 1.7%.

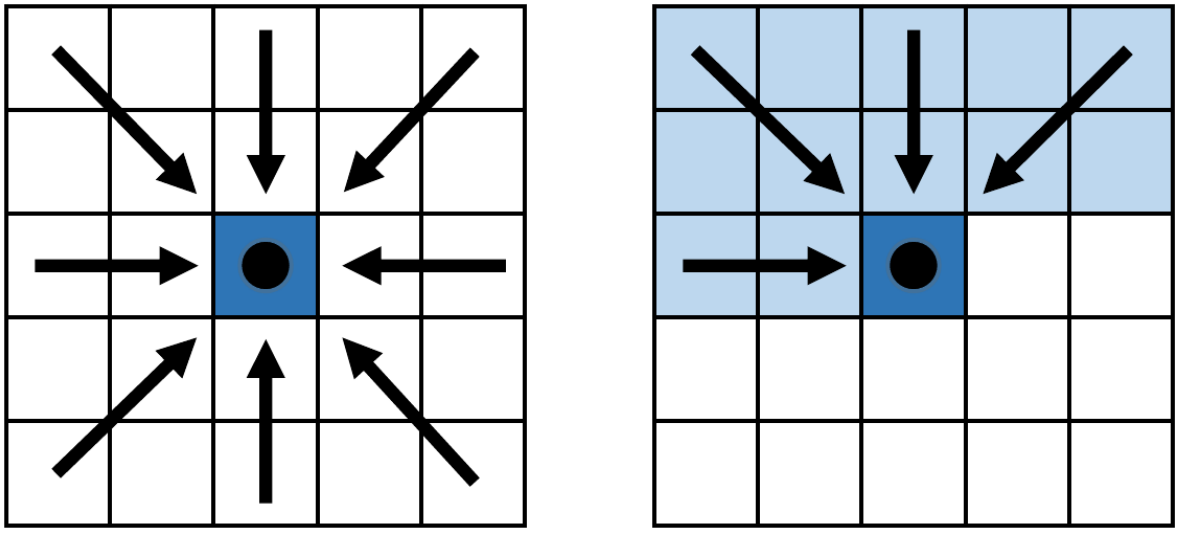


Figure 22: 8 paths aggregation vs. 4 paths aggregation.

GPU

General purpose computation on a GPU (GPGPU) allows for a massively parallelized computation of the SGM algorithm without major changes to the stategy of the algorithm. However, independent processing steps are to be identified in order to optimaly parallelize the algorithm.

B.1.6 Evaluation of the guideline in reference applications

Evaluation of the guideline is done as part of the UAV use case.

B.1.7 References

[1] Hirschmueller H. Stereo processing by semiglobal matching and mutual information. IEEE Transactions on Pattern Analysis and Machine Intelligence, 30(2), pp. 328-341, 2008.

[2] Banz, C., Hesselbarth, S., Flatt, H., Blume, H. and Pirsch, P. Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga- implementation. In: Proc. International Conference on Embedded Computer Systems, pp. 93–101, 2010.

[3] Barry, A. J., Oleynikova, H., Honegger, D., Pollefeys, M. and Tedrake, R. Fpga vs pushbroom stereo vision for UAVs. In: IROS Workshop on Vision-based Control and Navigation of Small Lightweight UAVs, 2015.

[4] Hofmann, J., Korinth, J. and Koch, A. A scalable high-performance hardware architecture for real-time stereo vision by semi-global matching. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 27–35, 2016.

[5] Li, Y., Li, Z., Yang, C., Zhong, W. and Chen, S. High throughput hardware architecture for accurate semi-global matching. Integration, 2017.

[6] Spangenberg, R., Langner, T., Adfeldt, S. and Rojas, R. Large scale semi-global matching on the CPU. In: Proc. IEEE Intelligent Vehicles Symposium, pp. 195–201, 2014.

[7] Banz, C., Blume, H. and Pirsch, P. Real-time semi-global matching disparity estimation on the GPU. In: Proc. IEEE International Conference on Computer Vision Workshops, pp. 514–521, 2011.

[8] Gehrig, S. K. and Rabe, C. Real-time semi-global matching on the CPU. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 85–92, 2010.

[9] Haller, I. and Nedevschi, S. GPU optimization of the sgm stereo algorithm. In: Proc. IEEE International Conference on Intelligent Computer Communication and Processing, pp. 197–202, 2010.

B.1.8 Related guidelines

- B.24 How to optimize SGM for GPGPU
- B.23 Remove recursion when aiming to parallelize code

B.2 Choosing a Real-time Operating System

B.2.1 Guideline Information

Guideline Responsible	Antonio Paolillo, HIPPEROS
Guideline Reviewer	Philippe Millet, Thales
Guideline Audience	OS designers, System architect
Guideline Expertise	OS designers
Guideline Keywords	Component selection

B.2.2 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 based on modular micro kernel architecture.

B.2.3 Insights that led to the guideline

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

B.2.4 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.

B.2.5 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.

B.2.6 Evaluation of the guideline in reference applications

The operating system is tested on the medical use case and necessary to copy with the requirement to never lose any frame. The RTOS ensure that the tasks are always executed the way it is defined at design time.

B.3 Turn off the FPGA device to save power if the latencies allow it

B.3.1 Guideline Information

Guideline Responsible	Björn Gottschall, NTNU
Guideline Reviewer	Timoteo García Bertoa, Sundance
Guideline Audience	System architect
Guideline Expertise	HW designers
Guideline Keywords	Energy efficiency, FPGA

B.3.2 Guideline advice

Turning SRAM-based FPGA devices off can save a significant amount of energy, but adds latencies from the start-up process and required reconfiguration. To avoid depleting energy in batteries, the platform should be switched off for longer idle times.

B.3.3 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 necessarily transfer to SRAM-based FPGA designs since they have to be reconfigured when powered on. Reconfiguration consumes extra power and adds additional latencies. The trade-offs are different for FPGAs that use non-volatile memory to store configuration data.

B.3.4 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.

B.3.5 Instantiation of the recommended implementation method in the reference platform

Not all vendors provide power measurement capability on-board. This is true for the for the EMC2DP, a predecessor for the reference platform. NTNU has constructed 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.

B.3.6 Evaluation of the guideline in reference applications

Case	Power [W]	Time [mS]	Energy [J]
WFI	4.95719	10000	49.5719
PL-SD	5.05152	756.087	3.81939
PL	5.20629	17.040	0.08872

Table 7: Measuring power and energy for different FPGA configuration scenarios.

Before comparing the energy consumption between idle and PL (Programmable Logic, FPGA) reconfiguration, it is required to define the idle state of the processing system (APU and/or RPU). Idle is considered a state from which the platform can wake up itself by any event. The least amount of energy is drawn by holding the processing system in the WFI (wait for interrupt) state as shown in the table above. Performing busy waiting or polling does consume more energy and is therefore not recommended as an idle state in an energy-critical environment.

Reconfiguration of the PL from a bitstream residing on the SD-Card takes 756 ms and consume 5.05 Watt as seen in the Table 7 (PL-SD), which is roughly 0.1 watts more than the idle state. Measurements of the PL reconfiguration from an in-memory prepared bitstream have shown that the limiting factor is the slow SD-Card interface. Only 2.3% of the energy required for reconfiguration from the SD-card is invested in the actual PL reconfiguration. Thus, using a faster interface than SDHC can save a reasonable amount of energy.

The measurements show that switching off the platform saves a significant amount of energy and that PL reconfiguration not only consumes slightly more energy than idling but is also a very fast process, depending on the interface used. However, temporarily switching off the platform adds latencies from the startup process that needs to be considered. Depending on the application and required reaction speeds the latencies added by PL reconfiguration alone could be negligible.

B.4 Move Data Processing as Close to the Sensor as Possible

B.4.1 Guideline Information

Guideline Responsible	Lester Kalms, TUD
Guideline Reviewer	Magnus Peterson, Synective Labs
Guideline Audience	System architects, Application developers
Guideline Expertise	Toolchain designers
Guideline Keywords	Energy efficiency, Sensor data

B.4.2 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.

B.4.3 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.

B.4.4 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.

B.4.5 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).

B.4.6 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 data processing as close as possible to the sensor – combined with simple optimizations – can improve your system.

B.5 IP-Core for an Image Processing Platform using FPGAs

B.5.1 Guideline Information

Guideline Responsible	Flemming Christensen, Sundance
Guideline Reviewer	Magnus Jahre, NTNU
Guideline Audience	HW designers, System architect
Guideline Expertise	HW designers
Guideline Keywords	IP Cores; Platform selection; OpenVX; OpenCL; MathWorks, LabVIEW-FPGA; HLS; 'C-to-VHDL'

B.5.2 Guideline advice

The choice of an FPGA for an Image Processing system has benefits in terms of flexible Input/Output, as can implement the any camera standard found in Table 1 of the TULIPP D1.1 Reference Platform[3].

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 typically comes as an IP-Core, but it uses 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]

B.5.3 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 an ARM-CPU integrated into the FPGA by default, so the IP Cores we refer to here will be for the FPGA fabric.

B.5.4 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: * 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.

[3] http://tulipp.eu/wp-content/uploads/2016/12/d1.1_reference_platform_v1_20161215.pdf, page 35

Publisher, like Wiley[4], has more than 50 titles on the subject of image processing and the OpenCV[5] community has thousands of programs that are BSD licensed and free.

* What about 'hardware-land'? Well, not many, as highly specialized. The best place is OpenCores[6]. This site has 300000+ 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.
* The handful of FPGA vendors that sells devices to the open market have their own 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[7], so difficult to complain. Never seems to support the latest vendor specific tool-chain, hence not very useful. * Independent vendor IP-Cores, like BarcoSilix'[8] 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 * 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 tool-chain and enables porting of current applications that can be tested on a CPU and moved to FPGA. * The high-level Model programming of FPGA is also possible, but expensive. Tools like MathWork's Simulink[9] [vendor agnostic] and NI's LabVIEW[10] FPGA [vendor specific] can create executable code that can be ported to the FPGA found on TULIPP Starter Kit.

B.5.5 Instantiation of the recommended implementation method in the reference platform

A good and short read is TULIPP Platform Concept[11] for an overview of a process to select an IP-Core. 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.

[4] https://www.wiley.com/en-gb/General+%26+Introductory+Electrical+%26+Electronics+Engineering/Image+and+Video+Processing-c-EEJ1
[5] http://opencv.org/
[6] https://opencores.org/
[7] https://www.xilinx.com/products/intellectual-property/ef-di-vid-img-ip-pack.html
[8] https://www.altera.com/solutions/partners/partner-profile/barco-silex.html
[9] https://uk.mathworks.com/solutions/fpga-design.html
[10] http://www.ni.com/labview/fpga/
[11] http://tulipp.eu/tulipp-platform-concept/

B.5.6 Evaluation of the guideline in reference applications

Some of the TULIPP "Starter Kit" will contain a few examples of IP-Cores that has been selected or developed to support our Use-Cases and these have been shared on Hackaday IP[12] and the Medical[13]/ADAS[14] uses the HDMI Out[15] IP-Core and the UAV[16] Use-Case also uses the Camera IP[17] to get video stream from a camera. One of the TULIPP "Starter Kit", called "AGRI"[18] has a number of IP-Cores that are shared with the community

[12] https://hackaday.io/
[13] tulipp.eu/medical-x-ray-imaging
[14] tulipp.eu/advanced-driver-assistance
[15] https://hackaday.io/project/28240-zynq-ultrascale-hdmi-out
[16] http://tulipp.eu/surveillance-and-rescue-uavs/
[17] https://hackaday.io/project/27372-cameralink-on-the-zynq-based-tulipp-platform
[18] https://www.slideshare.net/sundancedotcom/tulipp-starter-kit-agri

B.6 Reordering Loops can Reduce Bandwidth Requirements

B.6.1 Guideline Information

Guideline Responsible	Magnus Peterson, Synective Labs
Guideline Reviewer	Ariel Podlubne, TUD
Guideline Audience	Toolchain designers, Application developers
Guideline Expertise	Application developer, FPGA programmer
Guideline Keywords	Code optimization

B.6.2 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 (BRAM etc, as opposed to external DDR)

B.6.3 Insights that led to the guideline

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

B.6.4 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.

B.6.5 Instantiation of the recommended implementation method in the reference platform

This guideline is addressedin most of the use case implementations.

B.6.6 Evaluation of the guideline in reference applications

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

B.7 Upgrading to newer parts/FPGA architectures

B.7.1 Guideline Information

Guideline Responsible	Timoteo García Bertoa, Sundance
Guideline Reviewer	Igor Tchouchenkov, Fraunhofer
Guideline Audience	HW designers
Guideline Expertise	HW designers
Guideline Keywords	hardware, upgrade, fpga architecture

B.7.2 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.

B.7.3 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.

B.7.4 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.

A way to verify changes due to hardware updates, is creating spreadsheets, locating the different FPGA pins for the different signals available. Displaying FPGA pin, I/O Standard, Voltage range, termination (if existing), etc. is a very good approach to spot errors or possible incompatibilities between hardware parts, to determine a solution previous to software testing, avoiding damaging the hardware. The spreadsheets should be created from the original circuit schematics, unless they are already provided by the hardware supplier.

B.7.5 Instantiation of the recommended implementation method in the reference platform

A spreadsheet for the EMC2-DP carrier board was created by Sundance, displaying the pinout for the different signals, showing in columns the modules used along the project, being Zynq Ultrascale+ the last goal. Also, FMC boards (HDMI in/out from Avnet, Camera Link from Sundance DSP) were added, to study the I/O standards used for critical interfaces where the polarity of the differential pairs might cause trouble if they appear to be swapped.

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.

B.7.6 Evaluation of the guideline in reference applications

Progress was made throughout the project, where demos were running successfully despite the hardware upgrades.

The project started with Zynq 7000 family and then upgraded toward the MPSoC family with zu3 and later with zu4. Problems with polarities and voltage incompatibilities were solved using specific I/O standards, or swapping pairs by software using VHDL/Constraints in Vivado.

B.7.7 References

[1] SoC Carrier, EMC2-DP, Sundance Multiprocessor Technology LTD, 2016, https://www.sundance.technology/som-cariers/pc104-boards/emc2-dp/
[2] Zynq 7000, TE0715-15, Trenz Electronic, 2016, https://shop.trenz-electronic.de/en/te0715-04-15-1i-soc-micromodule-with-xilinx-zynq-xc7z015-1clg485i-ind.-temp.-range
[3] Zynq 7000, TE0715-30, Trenz Electronic, 2016, https://shop.trenz-electronic.de/en/te0715-04-30-1c-soc-micromodule-with-xilinx-zynq-xc7z030-1sbg485c-com.-temp.-range
[4] Zynq Ultrascale+, TE0820-3EG, Trenz Electronic, 2017, https://shop.trenz-electronic.de/en/te0820-03-03eg-1eb-mpsoc-module-with-xilinx-zynq-ultrascale-zu3eg-

1e-4-gbyte-ddr4-sdram-4-x-5-cm?c=242
[5] Zynq Ultrascale+, TE0820-4EV, Trenz Electronic, 2018, https://shop.trenz-electronic.de/en/te0820-03-04ev-1ea-mpsoc-module-with-xilinx-zynq-ultrascale-zu4ev-1e-2-gbyte-ddr4-sdram-4-x-5-cm?c=242
[6] FMC module, FMC-CL, Sundance DSP, 2017, https://www.sundance.technology/system-on-modules-som/fmc-modules/io-modules/fmc-cl/
[7] FMC module, AES-FMC-HDMI-CAM-G, Avnet, 2015, https://www.avnet.com/shop/us/products/avnet-engineering-services/aes-fmc-hdmi-cam-g-3074457345635221625/

B.8 Optimize nested loops with early exit by measuring the number of unconditional iterations

B.8.1 Guideline Information

Guideline Responsible	Magnus Peterson, Synective Labs
Guideline Reviewer	Philippe Millet, Thales
Guideline Audience	Application/algorithm developers
Guideline Expertise	Application/algorithm developers
Guideline Keywords	Code optimization, FPGA

B.8.2 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.

B.8.3 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.

B.8.4 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.

B.8.5 Instantiation of the recommended implementation method in the reference platform

This guideline is used in the classifier implementation in the automotive use-case.

B.8.6 Evaluation of the guideline in reference applications

With the relation between the inner and the outer loops that the classifier implementation has, taking this guideline into consideration had significant impact on the processing speed.

B.9 Source Code Organization

B.9.1 Guideline Information

Guideline Responsible	Asbjørn Djupdal, NTNU
Guideline Reviewer	Ahmed Sadek, TUD
Guideline Audience	Application Developer
Guideline Expertise	Toolchain Designer
Guideline Keywords	Toolchain

B.9.2 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. This consideration must be done while executing the generic development process described in D4.4 chapter 1.

B.9.3 Insights that led to the guideline

Experience with Xilinx SDSoC led to this guideline.

B.9.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

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.

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.

B.9.5 Instantiation of the recommended implementation method in the reference platform

All applications will benefit from this guideline, and the TULIPP use cases are examples of this.

B.9.6 Evaluation of the guideline in reference applications

Build times are hugely dependent on application and build computer specifications. However, an example can give an indication of the importance of the guideline. An LKOF image processing application needs 24 minutes to be built on a typical desktop PC when touching one of the HW files. If only a SW file is touched, the application builds in 40 seconds. This demonstrates that keeping SW and HW code in separate files is important for build efficiency.

B.10 Automating the Toolchain

B.10.1 Guideline Information

Guideline Responsible	Asbjørn Djupdal, NTNU
Guideline Reviewer	Antonio Paolillo, HIPPEROS
Guideline Audience	Application Developer
Guideline Expertise	Toolchain Designer
Guideline Keywords	Toolchain

B.10.2 Guideline advice

Automate as much as practically possible with scripts.

B.10.3 Insights that led to the guideline

Experience with different projects for different IDEs has shown the usefulness of using command line build tools. Also, running large numbers of experiments has shown the importance of scripting for reducing error-prone human input during the experiment runs.

B.10.4 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. Time should be spent to learn these capabilities, and use them when doing development following the generic development process explained in D4.4 chapter 1.

Examples of operations suited for scripts:

1. Automating the build process. Using GNU makefiles or CMake can be more dependable and easier in the long run, compared to specifying build configurations inside some unstandardised GUI tool. This is especially true when several developers work on the same project. Specifying the build process in this way is also a requirement for other automated operations, such as end-to-end automated testing. This way of building is also much more suited for version control systems.
2. Repetitive and/or error prone operations. Example: Regression testing a new build on the target platform, when interactive debugging capabilities are not needed.
3. Complex operations where the steps are unlikely to be remembered to the next time it is required. Example: Running an experiment on the target platform and creating reports from program runs

4. 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.

B.10.5 Instantiation of the recommended implementation method in the reference platform

The analysis tool (D4.4) has been realised, partly due to the scripting capabilities of the GNU and Xilinx development tools. Makefiles are generated automatically which control the build process and ties all tools together. Most functionality of the tools provided by TULIPP are available from the command line, and is thus scriptable.

B.10.6 Evaluation of the guideline in reference applications

Although development time when using tools such as CMake is perceived to be reduced significantly, it is difficult to quantify, or prove, this advantage. The same goes for other usages of scripting, such as automated tests. However, most experienced developers would probably agree that command line tools and scripting enhances productivity. The analysis tool shows that the possibility to control tools through makefiles can be very useful.

B.11 Timestamp ADC samples promptly

B.11.1 Guideline Information

Guideline Responsible	Björn Gottschall, NTNU
Guideline Reviewer	Magnus Peterson, Synective Labs
Guideline Audience	System architect
Guideline Expertise	System architect
Guideline Keywords	Sampling, performance analysis

B.11.2 Guideline advice

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

B.11.3 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.

B.11.4 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 them 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.

B.11.5 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.

B.11.6 Evaluation of the guideline in reference applications

Profiling applications intrusively by requesting samples from an external ADC inevitably adds latencies for the communication. This latency can vary widely depending on the used interface and the communication stack. Lynsyn supports this type of sampling via an I2C

interface, which adds ∼300 us latency for the request of one sample. To overcome this issue, Lynsyn provides non-intrusive timestamped sampling, adding no sampling latencies from any communications due to local buffering on the ADC. The host collects the buffered data and does not interfere with the actual sampling process or the profiled application running on the target. In this way, the acquired samples represent the analog signal as good as possible.

B.12 Save implementation time by delegating customizations to the hardware vendor

B.12.1 Guideline Information

Guideline Responsible	Magnus Jahre, NTNU
Guideline Reviewer	Michael Grinberg, Fraunhofer
Guideline Audience	Application developers
Guideline Expertise	HW designers
Guideline Keywords	Component selection, System optimisation

B.12.2 Guideline advice

Save implementation time by delegating customizations to the hardware vendor.

B.12.3 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 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.

B.12.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

If a 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.

B.12.5 Instantiation of the recommended implementation method in the reference platform

The TULIPP platform uses system-on-module products from Trenz Electronics. Thus, the detailed implementation of the interfacing of the Xilinx UltraScale+ MPSoC to for instance external memory has been delegated to Trenz Electronics.

B.12.6 Evaluation of the guideline in reference applications

The PCB-level interfacing of a complex component such as the UltraScale+ is complex and error-prone. Conservatively, we assume that developing this subsystem in-house would take on the order of 3 to 6 person months. At NTNU, a rough cost estimate would be between 20 to 40 KEUR depending on how long it takes and the salary of the person doing the design. In contrast, a Trenz module costs a few hundred euros (including component costs).

B.13 Selecting PSU to supply the chosen hardware

B.13.1 Guideline Information

Guideline Responsible	Timoteo García Bertoa, Sundance
Guideline Reviewer	Igor Tchouchenkov, Fraunhofer
Guideline Audience	HW designers, System architect
Guideline Expertise	HW designers
Guideline Keywords	Power, PSU, voltage, load, hardware

B.13.2 Guideline advice

The user should choose the appropriate PSU when testing the corresponding application on the hardware, and appropriately wire it to the low-powered board. PC PSUs are not always providing the correct voltages, as there might be a voltage drop at the wiring, and voltage levels might not be true values at the load. This can mislead users to think the hardware has a problem.

B.13.3 Insights that led to the guideline

Some of the partners within the Tulipp project have faced the issue of finding “harware 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). PSUs require a minimum load impedance in order to supply the expected output voltage levels. Some PC PSUs are design with secondary transformers, for low power applications where the 5V rail draws small current, but it’s not always the case. Also, using a PSU that is thought for a PC motherboard, might not be using the VSENSE rail correctly, to provide true levels at the load, in this case the Tulipp board.

B.13.4 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 electronic instrument. If the voltage level is not enough, the user should check the output voltage provided by the PSU, and verify that there is no voltage loss at the wiring. This could be solved wiring VSENSE directly to the load. If there is no voltage loss, an additional load could be added, to increase the overall load impedance, ensuring the PSU is working efficiently as specified by the vendor.

B.13.5 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. In case of voltage loss, VSENSE could be wired to one of the 3.3V points on the Tulipp board. In case of not voltage loss, to discard PSU's efficiency, a hard disk could be connected through SATA, to increase the load impedance.

B.13.6 Evaluation of the guideline in reference applications

Some users had this issue in their testing process, and it was solved, as it can be seen in their reports:

- `http://support.tulipp.eu/viewtopic.php?f=8&t=20`
- `http://support.tulipp.eu/viewtopic.php?f=8&t=18`

B.14 Streaming accelerators should use on chip resources for communication

B.14.1 Guideline Information

Guideline Responsible	Lester Kalms, TUD
Guideline Reviewer	Magnus Jahre, NTNU
Guideline Audience	Application developers
Guideline Expertise	Toolchain designers
Guideline Keywords	Energy efficiency, Code optimization, FPGA

B.14.2 Guideline advice

Exchange data between streaming capable functions via on-chip resources by fusing the functions into one single accelerator if possible.

B.14.3 Insights that led to the guideline

There are different possibilities to communicate data between streaming capable functions. The first method would be to exchange data via main memory. This creates resource overhead, due to additional interfaces, DMAs and interconnection networks. Furthermore, the communication increases the latency and the bandwidth utilization. A second method takles these issues by streaming the data between two seperate accelerators. This can either be done in the Vivado block design or by automatized by the SDSoC tool. The last method would optimize this by implementing both functions into one accelerator, which would further reduce additional hardware resources needed for the communication and integration of both functions. Additionally, it is faster to implement the last method.

B.14.4 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.

B.14.5 Instantiation of the recommended implementation method in the reference platform

A more precise example is provided in the Image Processing Library (IPL), which uses the same methodology as in the example above. In the main.cpp file there is a function called “hwTestExampleApplication”.

B.14.6 Evaluation of the guideline in reference applications

We compared the off-chip and on-chip method with each other using the Image Processing Library for the example described above. For evaluation, we used the estimation results of the SDSoC tool using the zcu102 for 1080p images.

Resource	On-Chip	Off-Chip
DSP	2	2
BRAM	19	50
LUT	12258	29024
FF	13176	35556
Estimated Time	23.6 ms	66.2 ms

Table 8: Resource utilization of on-chip and off-chip implementation methods.

B.15 Integrating your image processing HDL code with the platform

B.15.1 Guideline Information

Guideline Responsible	Fatima Kishwar, Sundance
Guideline Reviewer	Magnus Jahre, NTNU
Guideline Audience	Toolchain designers, Application developers, System architects
Guideline Expertise	HW designers
Guideline Keywords	FPGA, System optimisation

B.15.2 Guideline advice

If you have a HDL code that can do part of the processing, you can integrate it with your SDSoC design via IP creation in Vivado.

B.15.3 Insights that led to the guideline

Certain applications need to use legacy HDL modules to interface with certain peripherals (where Xilinx IP's are not present) or interface third party IP's or have a legacy module written in HDL to be used in the project design.Feature-based Simultaneous Localization and Mapping (SLAM) algorithm have as the first processing step the feature detection and description task. The availability of HDL based implementations for some functions like Harris or FAST corner detection can be utilized within an SDSoC Project.

B.15.4 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 needs 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).

B.15.5 Instantiation of the recommended implementation method in the reference platform

Two kind of Cameras are used in TULIPP's projects, HDMI-based and Cameralink-based Camera. The combination of EMC2 and Avnet HDMI Input/Output FMC board was used in the Sobel and motion detection demo. Integrating HDL with this platform require finding-out the stream of pixels and the synchronization signals. Custom HDL code for the Avnet hardware module is provided to use with the board and needed to be integrated with the AXI

based design of the project; this was done using the IP integration tool present in the Vivado toolset.

B.15.6 Evaluation of the guideline in reference applications

This was tested both by Sundance and Hipperos during their demos implementation.

B.16 Use Hardware In the Loop HIL testing rather than wait for a final hardware

B.16.1 Guideline Information

Guideline Responsible	Sebastian Monka, Aleksej Buller, Fraunhofer IOSB
Guideline Reviewer	Timoteo García Bertoa, Sundance
Guideline Audience	HW designers, System architect
Guideline Expertise	HW designers, System architect
Guideline Keywords	testing, drones, project management, time management

B.16.2 Guideline advice

Use Hardware in the loop (HIL) testing with early prototypes rather then wait for an fully integrated hardware[1].

B.16.3 Insights that led to the guideline

How the drone (new Tulipp platform) communicates with the outside world please look at Guideline B.17.

B.16.4 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 your are not limited to the Weather or risks of drone crashes.

There are a couple of software tools that replicate a UAV in a computer (HIL [3]). All properties of the drone (communication, flight parameters) are displayed in the simulation. It is also possible to control a UAV in the simulation via a manual remote control (flight simulator).

It is correct to test the new software-algorithms not immediately on the drone, but in the simulation. A small mistake can have a big impact. On a land robot you can install a wireless emergency stop, but on a UAV an emergency stop function does not work in a fault condition. Made a mistake and the drone can fly away or crash. The big effort would have been for nothing.

With AIRSIM [2] the MAVLink communication between the Tulipp platform and the flight controller was tested. The algorithms for the video evaluation were tested in the QT Creator [4] with openCV library [5].

B.16.5 Instantiation of the recommended implementation method in the reference platform

Obstacle detection on the Tulip reference platform was implemented. The safe flight of the obstacles only with the help of the depth maps is currently not possible.

In the last phase of development, the “hardware in the loop” helps us very little, because we want to evaluate the pictures from the real world and deal with real obstacles.

B.16.6 Evaluation of the guideline in reference applications

We could not change a parameter for the altitude above ground. The drone in the simulation has been rising higher and higher. Software compilation in SDSoC took about 2.5 hours. There are many other unexplainable issues that sporadically appear and slow our progress.

B.16.7 References

[1] B.17 How to get EMC2 Board communicate with an UAV

[2] AIRSIM, `https://dev.px4.io/en/simulation/airsim.html`

[3] Hardware in the loop: Hardware-in-the-loop_simulation

[4] QT-Creator: Qt_Creator

[5] OpenCV Library, `https://opencv.org/`

B.17 How to get EMC2 Board communicate with an UAV

B.17.1 Guideline Information

Guideline Responsible	Sebastian Monka, Aleksej Buller, Fraunhofer IOSB
Guideline Reviewer	Paul Rodriguez, HIPPEROS
Guideline Audience	System architect
Guideline Expertise	HW designers, System architect
Guideline Keywords	UAV control, serial I/O, UART

B.17.2 Guideline advice

The most important step for getting the EMC2 Board to work with an UAV is to establish a connection. There are different steps to be achieved in order to get a proper running platform that can be used to develop the application and well connected to the development environment to get the best possible development experience.

B.17.3 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 important to describe the implementation of I/O ports through the whole stack of hardware and tools.

B.17.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

This guideline exhibits the need for standards to interconnect the hardware parts one to another but also to benefit from opensource software on top of well defined and widely used interfaces.

B.17.5 Instantiation of the recommended implementation method in the reference platform

This guideline is illustrated through the development of the UAV use case

Setting up in Vivado

First of all a Hardware Platform has to be built in Vivado where all the in/- and outputs are defined. 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 fastly generate your own linux distribution for testing your hardware. Please be aware that this distribution is not fully recommended for the UAV use case as it is not a real-time OS and will therefore not guaranty the deadline of the executed tasks. However, this operating system is widely used on the Zynq platform and it is valuable to describe it here. Note that the HIPPEROS operating system developed in Tulipp is also integrated in the SDSoC environment of Xilinx and will offer the same IO services as Petalinux.

Informations about how to setup a Petalinux Environment and how to use Petalinux can be found inside the Xilinx Documentations[19]. - 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 the project can be built. Petalinux creates different files which need to be combined into a BOOT.bin file and an image.ub file. These files need to be included into the BOOT folder of the SD-card that is connected to the Zynq and that contains the bootloader, the operating system and the applications. The SD-card can be used as a data storage device for the application.

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 off. If this is not the case

[19] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug1157-petalinux-tools-command-line-guide.pdf

probably there is something wrong with the generated boot files in Petalinux or Vivado. - unmount the SD-card from the workstation - plug it on the target board and switch the board on. - 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 baud rate of 115200. Note that depending on the workstation hardware and the Linux distribution, the USB might appear as another device name. To check that name, use “dmesg -w” command. When the USB is connected to the workstation, Linux puts a message that indicated that a new device is connected and gives the name of that device.

UAV Control

Our UAV is equipped with a Pixhawk 4 autopilot. This flight controller is an open source autopilot. The APM / Pixhawk platform is extremely flexible in terms of functionality as well as in hardware enhancements, firmware, and add-on software (for various OS) developed by a large developer community. With APM it is possible to control any kind of model, the most important variants are APM: Copter for Multicopter, APM: Tarpaulin for Surface Models and APM: Rover for Vehicles (also suitable for boats). Incidentally, the old names are ArduCopter, ArduPlane and ArduRover [6].

Our drone “speaks” MAVLink (messaging protocol).

To communicate with the Tulipp-Drone, an intermediary (translator) is needed. The Tulipp Platform transfers the image coordinates of the obstacle via USB (/dev/ttyUSB0) to the mediator. The mediator reads the current position of the UAV (GPS) via the autopilot (GPS-Sensor), calculates new waypoints (3D coordinates), encodes them in new Mavlink matches and sends Mavlink-Messages to the drone via a UART (/dev/ttyAMA0).

A role of the mediator takes over a Raspberry Pi [8], single-board computer. The conversion algorithms are programmed in Python [9][10].

The autopilot is configured via Mission Planner [11] for communication with Raspberry Pi via the telemetry interface.

B.17.6 Evaluation of the guideline in reference applications

This method was used to develop the UAV use case in the Tulipp project. It exhibits the basic interfaces required for the software stack used on top of the hardware platform.

Tulipp-Platform renewal, change to new FPGA

Since mid-2017 we are implementing our algorithms for obstacle detection using a stereo camera system on a new Tulipp platform with MPSoC module with Xilinx Zynq UltraScale + ZU3EG-1E from TRENZ-Electronic (TE0820) [4] on the carrier board EMC2-DV of Sundance Technology [1]. The hardware implementation on the FPGA has been adjusted accordingly in Vivado. Our stereo camera system is based on the hardware worked out for us by Synective

Labs (Block Diagram). To process the two video streams on the FPGA, a Cameralink frame grabber FMC board [2] is used. An operating system (e.g. Hipperos or petalinux) on our board is currently not installed. Our algorithms have been implemented with SDSoC in C/C++ [5]. The FPGA runs in standalone mode. Activation of the serial interface is similar to Zynq-7030.

B.17.7 References

[1] EMC2-DP Board, https://www.sundance.technology/som-cariers/pc104-boards/emc2-dp/

[2] FMC-CL Dual camera link LPC FMC, https://www.sundance.technology/system-on-modules-som/fmc-modules/io-modules/fmc-cl/

[3] PCI-104 Power Supply, https://www.sundance.com/product-range/sundance-products/pc104/smt1024/

[4] TRENZ TE0820, https://wiki.trenz-electronic.de/display/PD/TE0820+SDSoC

[5] Link to SDSoC Environment Platform Development Guide, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug1146-sdsoc-platform-development.pdf

[6] Ardupilot, http://ardupilot.org/copter/index.html

[7] MAVLink Protocol, https://mavlink.io/en/

[8] Raspberry Pi, https://www.raspberrypi.org/

[9] Pymavlink Libraries, https://mavlink.io/en/mavgen_python/

[10] Python, https://www.python.org/

[11] Mission Planner, http://ardupilot.org/planner/

B.18 Use an external constant voltage reference for the ADC instead of the internal reference based on positive supply voltage VDD

B.18.1 Guideline Information

Guideline Responsible	Asbjørn Djupdal, NTNU
Guideline Reviewer	Graeme Parker, Sundance
Guideline Audience	HW designers
Guideline Expertise	HW designers
Guideline Keywords	ADC, measurement

B.18.2 Guideline advice

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

B.18.3 Insights that led to the guideline

Positive supply voltage drifts with temperature and varies with load.

B.18.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Use constant voltage reference chips such as LM385 or LT1009. Check the accuracy of both the FPGA being used and the external reference chip. Xilinx Zynq-7000 and Artix-7 both have a internal references with stated accuracy of 1%. MicroSemi Fusion internal reference is better at 0.9%. Better still is the LT1009 at 0.2%. Some versions of the common LM385 are only 2.4%.

B.18.5 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.

B.18.6 Evaluation of the guideline in reference applications

The lynsyn board could have been used with the internal reference voltage derived from VDD. This would have made the ADC accuracy dependent on the accuracy of the power supply, and susceptible to noise from all devices drawing current from VDD. By using the LT1009, the accuracy is 0.2%.

B.19 Isolate designs to low-power domains to reduce power consumption

B.19.1 Guideline Information

Guideline Responsible	Björn Gottschall, NTNU
Guideline Reviewer	Philippe Millet, Thales
Guideline Audience	HW designers, System architects
Guideline Expertise	HW designers
Guideline Keywords	Energy efficiency

B.19.2 Guideline advice

Isolate functionality to power domains to reduce power consumption.

B.19.3 Insights that led to the guideline

High-performance embedded platforms commonly include different power domains that can be independently clock-gated or power-gated. With clock-gating, the selected domain is not clocked and therefore transistors do not switch – saving dynamic power. With power-gating the power supply of the domain is cut – saving both dynamic and static power. A disadvantage of power-gating is that state elements (e.g., registers, memory) loose their contents. Thus, application state must be restored when the domain is powered up.

B.19.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Constrain applications to work with low-power domains. Carefully trade the benefits of entering a power saving mode against the overheads of resuming application execution.

B.19.5 Instantiation of the recommended implementation method in the reference platform

Zynq UltraScale+ MPSoC has three power domains – full-power, low-power, and PL domain. 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. Powering individual domains requires separate power rails, currently not available on the Tulipp reference platform. The full-power domain includes the main Application Processing Unit (APU, ARM Cortex-A53), the DDR-Controller, GPU and high speed connectivity like PCIe. The low-power domain provides the Real-Time Processing Unit (RPU, ARM Cortex-R5), Configuration Security Unit (CSU), Platform Management Unit (PMU), system monitor and general low-speed connectivity like USB. The PL power domain includes only the reconfigurable FPGA fabric as Programmable Logic (PL).

B.19.6 Evaluation of the guideline in reference applications

The reference platform does not provide power-gating functionality due to shared power rails. Therefore, it is not possible to shutdown individual power domains. Also, the evaluation does not include the power consumption of the PL as it is highly application and implementation dependent. However, the Xilinx Power Estimator (XPE) provides a worst-case value of 5.5 Watts for the PL alone for a rather unrealistic 100% usage of all its resources. The following evaluation should give some insight on possible power savings on the Tulipp reference platform by putting the APU and RPU of the UltraScale+ platform in different power states as well as considering to completely power off the platform and loading the PL configuration on startup.

The following case studies have been done: 1. Having APU and RPU in reset state 2. Waiting for interrupt on APU and RPU 3. Compute intensive workload on APU and RPU 4. Memory intensive workload on APU and RPU 5. Waiting for interrupt on APU single core (all others are held in reset) 6. Programming PL from bitstream in memory 7. Programming PL from bitstream on SD-Card as seen in Table 9.

Case	Power [W]	Time [mS]	Energy [J]
1	3.01465	10000	30.1465
2	4.95719	10000	49.5719
3	5.66150	10000	56.6150
4	6.66712	10000	66.6712
5	5.02270	10000	50.2270
6	5.20629	17.040	0.08872
7	5.05152	756.087	3.81939

Table 9: Power consumption for different platform states.

The least power is drawn in case 1, in which the RPU and APU is held in reset, while the PL configuration is preserved. However, the processing system cannot wake up itself from this state and must be initialized externally (e.g. JTAG). In case 2 all processor cores are set to WFI (wait for interrupt), which draws less power than any application execution. In contrast case 5 shows a single core from the APU in WFI state which consumes surprisingly more power. Case 3 and 4 show worst-case power consumption on the processing system from testing compute-bound and memory-bound applications. In case 6, the PL is reconfigured by a prepared bitstream in the off chip memory, which takes only 17 milliseconds on a slightly higher power consumption than idle (WFI) with 5.2 Watts. Case 7 shows a reconfiguration process from SD-Card, as a bitstream is usually not prepared in the main memory after power on. This requires less power but significantly longer time – due to the slow SD-Card interface.

These measurements clearly show that a considerable amount of energy can be saved by temporarily switching off the reference platform and investing the energy required for PL reconfiguration. However, application latencies added through the startup process needs to be considered in the design process. Waking the platform using interrupts saves around 0.5 Watts, keeps the PL reconfiguration in place and guarantees fast reaction times. Thus, the

method of choice depends on the application requirements.

B.19.7 References

[1] Managing Power and Performance with the Zynq UltraScale+ MPSoC, `https://www.xilinx.com/support/documentation/white_papers/wp482-zu-pwr-perf.pdf`

B.20 Beware the parameters of external data/clocking when testing or debugging

B.20.1 Guideline Information

Guideline Responsible	Timoteo García Bertoa, Sundance
Guideline Reviewer	Magnus Peterson, Synective Labs
Guideline Audience	HW designers
Guideline Expertise	HW designers
Guideline Keywords	Debugging, testing

B.20.2 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.

B.20.3 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 to 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 behavior of these, is using an external clock, asynchronous to the signals, but easily 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.

Using this strategy in order to test has its risks, and the user must be aware of the trade-off before thinking that the design is not working appropriately, when in reality the external clocking is not being applied correctly.

B.20.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

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. 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.

B.20.5 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.

This guideline applies to FPGA debugging with a logic analyzer. It shows how the hardware interfaces must be made available to be able to correctly diagnose hardware problems. This is a recommendation for building hardware platform to always let some signals available for debug AND to be aware of the proper way to interpret the output when the hardware runs faster than the analyzer.

B.20.6 Evaluation of the guideline in reference applications

This solution was useful to debug a camera link demo, where the video data was being analysed in order to check the interface protocol, assuming the possible misalignment or delays an asynchronous clock might bring into the design. The user thought that it wasn't working, when the external clock was wrongly applied, leading to fake information or malfunctioning.

B.21 Make sure to correctly access streamed data from within accelerated function

B.21.1 Guideline Information

Guideline Responsible	Boitumelo Ruf, Fraunhofer
Guideline Reviewer	Philippe Millet, Thales
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	Code optimisation, FPGA

B.21.2 Guideline advice

When using streamed data as in or output to a hardware accelerated function, it is important to perform the correct number of a read/write operation, in order to avoid a buffer overflow. Conditional access to streamed data (less number of read operation than data available) is not allowed and will lead to undesired behaviour, such as a buffer overflow of the FIFO buffer.

B.21.3 Insights that led to the guideline

The stereo algorithm for the UAV use case requires a simultaneous input of two images. In order to test whether the input images were captured correctly a debug output was generated in which upper half of the left image and the lower half of the right image were displayed simultaneously on the screen.

The image data is streamed into the accelerated function. Initially the appropriate data stream was accessed conditionally depending on the current row index. Executing the code with synthesized HW functions led to a failure of the program, as not all image data was read from the input stream, resulting in a buffer overflow.

The remedial action is to access all input data within each loop iteration and perform a conditional variable selection.

B.21.4 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 --> buffer overflow
        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;
    }
  }
}
```

B.21.5 Instantiation of the recommended implementation method in the reference platform

Instantiation as part of the uav use case.

B.21.6 Evaluation of the guideline in reference applications

Even though the examples above deals with a test case to debug whether the input images are captured correctly, conditional access to data streams is done multiple times in the implementation of the uav use case.

In most cases some data of the input streams need to be buffered (either within a seperate buffer or within a FIFO that is large enough), as the data is required in later loop iterations.

B.21.7 References

[1] Vivado HLS User Guide, `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf`

B.22 Writing for HW

B.22.1 Guideline Information

Guideline Responsible	Asbjørn Djupdal, NTNU
Guideline Reviewer	Julian Haase, TUD
Guideline Audience	Application developer
Guideline Expertise	Toolchain designer
Guideline Keywords	Code Optimisation

B.22.2 Guideline advice

To keep the possibilities for HW/SW partitioning as open as possible, functions doing heavy calculations 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.

B.22.3 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.

B.22.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

High level synthesis (HLS) makes it possible to write HW modules to be executed on an FPGA using normal C code. This is one of the programming models mentioned in chapter 5 of D1.3. HLS 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 can not, 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 guidelines increases the possibility that your code will be HLS compatible. In addition, the HLS analysis capability of the analysis tool (D4.4) can be used at any stage to assess the HLS compatibility of any part of the application code.

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 fulfill 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.

B.22.5 Instantiation of the recommended implementation method in the reference platform

The LKOF image processing application has its main compute kernel written in two different versions: One optimized for running on the CPU and one optimized for HLS.

B.22.6 Evaluation of the guideline in reference applications

The LKOF image processing application demonstrates that the same optimisations can not be done for both running on the CPU and for HLS, and that the general advise of avoiding premature optimisations is valid also here. Instead of optimising early on, the LKOF application demonstrates that it is better to isolate the compute kernels early on and design them such that they are compatbile with both HLS and optimisations for running on the CPU. This makes sure that a future decision on optimizing for either HLS or CPU does not require a complete rewrite.

B.22.7 References

Xilinx User Guide UG902, `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf`

B.22.8 Related guidelines

- B.32 Ensure Dataflow for FPGAs by using HLS Inline Subfunctions
- B.33 Avoid using heavy libraries while writing source code for embedded systems

- B.34 Start from a pure C version of the application code before to optimize it for your embedded system

B.23 Remove recursion when aiming to parallelize code

B.23.1 Guideline Information

Guideline Responsible	Boitumelo Ruf, Fraunhofer
Guideline Reviewer	Paul Rodriguez, HIPPEROS
Guideline Audience	Application developers, Toolchain designers
Guideline Expertise	Application developers, Toolchain designers
Guideline Keywords	code optimization, GPGPU, FPGA, HLS

B.23.2 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.

B.23.3 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. 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.

B.23.4 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.

E.g.:

```
// tail-recursive
int factorial (int n, int acc = 1)
{
  if (n == 1)
    return acc;
  else
    return factorial(n - 1, acc * n);
}

// iterative
int factorial (int n)
{
  int acc = 1;
```

```
  for (; n > 1; --n)
    acc *= n;
  return acc;
}
```

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

B.23.5 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 border 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.

B.23.6 Evaluation of the guideline in reference applications

As APIs such as CUDA, OpenCL or Vivado HLS do not support recursive function calls, this guideline was only evaluated w.r.t. to the reference implementation. The reference implementation was run single-threaded on the CPU.

The removal of recursion in the path traversal leads to an increase of performance by a factor of 10.

B.23.7 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.

B.23.8 Related guidelines

- B.24 How to optimize SGM for GPGPU

B.24 How to optimize SGM for GPGPU

B.24.1 Guideline Information

Guideline Responsible	Boitumelo Ruf, Fraunhofer
Guideline Reviewer	Magnus Peterson, Synective Labs
Guideline Audience	Application developers, Toolchain designers
Guideline Expertise	Application developers, Toolchain designers
Guideline Keywords	code optimization, SGM, GPGPU

B.24.2 Guideline advice

In order to efficiently map the path aggregation of the semi-global-matching (SGM) 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.

B.24.3 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 as described in B.23.

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

B.24.4 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.

As a more generic view, this recommendation explains, through an example, how to modify an algorithm so that it will benefit from the underlying hardware.

B.24.5 Instantiation of the recommended implementation method in the reference platform

As a recommended implementation method, the algorithm is almost always develop without any knowledge of the final target used in the product. As the algorithm is often developed using high abstraction languages like Matlab or relying on state-of-the-art libraries like openCV, there is always a necessary step for the adaptation of that algorithm to the selected chip that will run it on the embedded system.

During this implementation, it often occurs that the algorithm needs to be adapted to use the full performance offered by this embedded target, depending on the accelerators available on the target. Therefore the first step is to understand the architecture of the accelerator (in this example a GPGPU with SIMD behaviour) and to reshape the code to cope with the architecture. Here it was necessary to cut the dependency between the data to allow the GPGPU to execute several instances of the same thread in parallel on different data.

B.24.6 Evaluation of the guideline in reference applications

The instantiation of this guideline was part of the GPGPU implementation of the UAV use case.

B.24.7 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.

B.24.8 Related guidelines

- B.23 Remove recursion when aiming to parallelize code

B.25 Save Intermediate Storage with Dataflow Regions

B.25.1 Guideline Information

Guideline Responsible	Asbjørn Djupdal, NTNU
Guideline Reviewer	Paul Rodriguez, HIPPEROS
Guideline Audience	System architect, application designer
Guideline Expertise	Toolchain designer
Guideline Keywords	Toolchain, FPGA, Code optimisation

B.25.2 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.

B.25.3 Insights that led to the guideline

Experimentation with Xilinx HLS examples.

B.25.4 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).

B.25.5 Instantiation of the recommended implementation method in the reference platform

The analysis tool includes special support for streaming applications. Streaming applications can be optimised with help from the automatic design space exploration (DSE) feature of the analysis tool. See section 2.3.1 in D4.4.

B.25.6 Evaluation of the guideline in reference applications

The LKOF image processing application will not fit in the FPGA fabric of the Zynq ultrascale if dataflow is not used. Intermediate buffers will be too large for the internal FPGA BRAM.

B.25.7 References

- Xilinx UG902 (Vivado HLS), `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf`

B.26 Use conditional branching carefully

B.26.1 Guideline Information

Guideline Responsible	Boitumelo Ruf, Fraunhofer
Guideline Reviewer	Magnus Jahre, NTNU
Guideline Audience	Application developers
Guideline Expertise	Hardware designers, System architects
Guideline Keywords	Code optimization, GPU, FPGA

B.26.2 Guideline advice

Conditional branching such as if-then-else is vital to most image processing applications, e.g. in finding maximum similarity between pixels or handling image boarders when filters exceed the dimensions of the image.

In terms of processing speed and performance overhead the use of conditional branching on CPUs and FPGAs is cheap. However, if the HLS tool cannot group branches, i.e. if branches are likely to diverge, the use of conditional branching can result in a resource overhead when optimizing code for FPGAs.

When leveraging the processing power of GPUs by GPGPU (general computation on a GPU), condition branching is to be used with caution. If branches diverge within a warp, i.e. if some evaluate to true and others to false, the instructions are executed twice, resulting in a processing overhead. [1][2]

B.26.3 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 GPUs, is able to only process the branch for which the conditional directive resolved to true.

FPGAs can also cope well with conditional branching in terms of processing speed, as HLS will create different paths for each conditional branch. However if the branches cannot be grouped efficiently the use of many conditional branches leads to a resource overhead on FPGAs

In order to achieve great parallelism and high data throughput, GPUs 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. The GPGPU programming model calls this paradigm Single-Instruction-Multiple-Threads (SIMT) which is similar to Single-Instruction-Multiple-Data (SIMD). This SIMT processing requires that all threads within in one warp (a group of threads running on one processor, sharing resources) 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. Furthermore, a divergence of the branches, which occurs when the conditional branch evaluates to true for some threads of the warp and for others to false, will result in processing overhead as the instructions are executed twice. See [1] and [2] for more information.

B.26.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Avoid conditional branching with possibly divergent branches. Use multiple loops to perform different operations in different areas of the image.

When accelerating code with GPGPUs instantiate different kernels instead of using if-then-else statements for image areas which need specific processing.

B.26.5 Instantiation of the recommended implementation method in the reference platform

This method is actually true for almost all accelerators and particularly with GPGPUs and FPGAs. Accelerators are often based on long pipeline chains and can manage big chunks of data with less hardware involved than standard CPUs. This must particularity be taken into account during the development of the algorithm as branches will cut the execution pipeline and will also have effects on the data to be served to the application and therefore their distribution in the system.

B.26.6 Evaluation of the guideline in reference applications

There was no evaluation done as part of the Tulipp project as the guideline is common practice when employing GPGPU. However, the authors of [3] did a thorough evaluation on the effect of divergent threads. However, the Tulipp use case followed this method for the development of their application on GPGPU and FPGA.

B.26.7 References

[1] CUDA C Programming Guide, `http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf`
[2] Efficient control flow restructuring for GPUs, `https://ieeexplore.ieee.org/abstract/`

document/7568315/
[3] Benchmarking the Cost of Thread Divergence in CUDA, https://arxiv.org/pdf/1504.01650.pdf

B.27 Do not use floating point computation on FPGA

B.27.1 Guideline Information

Guideline Responsible	Boitumelo Ruf, Fraunhofer
Guideline Reviewer	Magnus Jahre, NTNU
Guideline Audience	Application developers
Guideline Expertise	HW designers, System architects
Guideline Keywords	Code optimization, FPGA

B.27.2 Guideline advice

Using floating point computation on FPGAs without hard floating point units [1] requires multiple loop iterations for basic arithmetic operations. Hence, in order to achieve high data throughput avoid floating point operations on such architectures.

Note that even if a FPGA has hard floating point units, the number of these units is limited, which might result in the same issue.

B.27.3 Insights that led to the guideline

Using floating point datatypes in the implementation of the stereo algorithm for the uav use case led to low throughput after synthesizing the code with HLS. The Xilinx Zynq, which is part of the Tulipp platform, does not provide hard floating point units. Thus the HLS cores could not be pipelined properly, as mupltiple clock cycles were needed to perform a basic arithmetic operation on the floating point values.

B.27.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

If no hard floating point unit is available the developer has few options:

- If a high throughput is required, i.e. a low inition interval when pipelining the computation, the developer should try to get rid of any floating point computation.
- If floating point conputation is required, the pipeline initiation interval is to be increased in order to account for multiple clock cycles which are required for the floating point computation.

B.27.5 Instantiation of the recommended implementation method in the reference platform

Instantiation as part of FPGA implementation for the UAV use case. As high throughput is needed, the algorithm was adjusted in such a way that no flaoting point compuation is required.

B.27.6 Evaluation of the guideline in reference applications

Without the use of floating point operations an initiation interval (II) of one was achieved for the processing of each pixel. That is, a new pixel processed with each clock cycle. With floating point operation, HLS could only achieve an II of 4, resulting in a low throughput.

B.27.7 References

[1] Floating Point FPGA, https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/po/bg-floating-point-fpga.pdf

[2] Vivado HLS User Guide, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf

B.28 Avoid LibTIFF library and use raw image format

B.28.1 Guideline Information

Guideline Responsible	Alvin Sashala Naik, Thales
Guideline Reviewer	Michael Grinberg, Fraunhofer
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	Image decompression, hardware optimisation

B.28.2 Guideline advice

Avoid complex data structure (TIFF) and use raw image format instead.

B.28.3 Insights that led to the guideline

While we were porting the medical use case on the Tegra family processors, we got difficulties to integrate LibTIFF library because the memory resources required by the library while doing image compression and decompression were way too much compared to the resourced available on the target.

Another problem comes with data manipulation. When the format is too complex, it is difficult to modify the way filters are implemented and to optimise the source code for a given processor.

B.28.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Use simple data format. Compression will not free any RAM, on the contrary it uses more space because the processor needs both the compressed format for storage and the uncompressed format for manipulation. Doing compression and decompression will slow down the processing application. Compression is only effective for data storage or while transmitting data on a network to save bandwidth. Using simpler data format will allow easier manipulation of the data and will also make it easier to port the application on embedded targets.

We advice to use either PPM/PGM images format or hexdump format because the structure is similar to a matrix of pixels.

When using a live image sensor (i.e. a camera), we also advice to work on the raw data flow from the image sensor, for you would have to decompress the video prior to using it which costs CPU time. If you need compression on the data line because the bandwidth is not high enough, we advice to use hardware IPs to do the decompression because it will be the best option to save power consumption while also saving the bandwidth.

B.28.5 Instantiation of the recommended implementation method in the reference platform

The reference platform *per se* does not include any software libraries for image compression or decompression. However, depending on the chosen hardware components, hardware accelerators might be available in the platform. For instance, on the Zynq SoC, one may include IPs on the FPGA fabric to do data compression and decompression. Such an IP will be available for the application through the operating system interfaces of the driver that takes care of the FPGA of the Zynq from the ARM processor. The management of the IP itself has to be taken into account from the application code.

B.28.6 Evaluation of the guideline in reference applications

The medical use case was processing TIFF images. We modified the source code and changed the reference images format to PGM. This allowed to get ride of the LibTIFF library and to lower the memory usage of the application.

B.28.7 Related guidelines

- B.33 Avoid using heavy libraries while writing source code for embedded systems

B.29 Use EMVA1288 to compare cameras

B.29.1 Guideline Information

Guideline Responsible	Magnus Jahre, NTNU
Guideline Reviewer	Flemming Christensen, Sundance
Guideline Audience	System architect
Guideline Expertise	HW designers
Guideline Keywords	Component selection

B.29.2 Guideline advice

Use metrics from the EMVA1288 standard to compare cameras.

B.29.3 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.

B.29.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Use EMVA1288[20] metrics in datasheets to compare cameras. Avoid vendors that supply cameras without EMVA1288 or other standard comparison metrics.

B.29.5 Instantiation of the recommended implementation method in the reference platform

Camera selection is outside of the scope of the TULIPP reference platform. Any camera that supports the CameraLink standard will be compatible.

B.29.6 Evaluation of the guideline in reference applications

The guideline has not been evaluated since camera selection is outside of the scope of the TULIPP reference platform.

[20] https://www.emva.org/standards-technology/emva-1288/

B.30 Low latency image processing over TCP IP data stream

B.30.1 Guideline Information

Guideline Responsible	Alvin Sashala Naik, Thales
Guideline Reviewer	Paul Rodriguez, HIPPEROS
Guideline Audience	System architect
Guideline Expertise	Application developers, System architect
Guideline Keywords	TCP/IP, Gstreamer, OpenCV, FFMPEG

B.30.2 Guideline advice

Use low-latency IP stream for sensor input and image processing rather than too generic libraries.

B.30.3 Insights that led to the guideline

Integration constraints from Thales Electron Devices (TED) to provide short (<70ms) and guaranteed latency between the image capture and its display.

B.30.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

In order to transfer a video stream over Ethernet, several protocols are available but some of them are particularly well designed to offer the low-latency required by the Tulipp use cases.

On top of the protocol, the library that processes the images and connect to the communication medium also has an impact on the latency and must be correctly chosen and tuned.

Even though largely used in the image processing domains, OpenCV is not a good fit for embedded targets as its memory footprint is very large and is implemented for general purpose processors and GPUs.

One approach that provides low-latency is running the GStreamer library pipeline. The processing can then be done through OpenVX with zero-copy between the processing functions. The output can then again be done using GStreamer.

More importantly, one can use circular buffers to overcome the drawback of using UDP as a transport protocol for the video. When the application can afford to drop frames, the circular buffers can be flushed whenever needed.

RTSP is a standard used to transport control over video streams on the internet, e.g. it offers a way to start/stop or pause a video stream. This standard comes with the RTP protocol,

the standard that transport the video it-self. They have been developed to transport real-time video streams and is widely used for serving videos on Internet, e.g. TV over Internet.

GigE Vision is a protocol developed for industrial applications to transmit high-speed video over Ethernet, relying on UDP.

All these protocols, even if they are meant to be real-time, are relying on the Internet Protocol Stack and the Ethernet drivers.

Today, many companies are using standard NICs to transfer image data from the camera into the host memory or directly to a display controller with low latency and jitter. However, when image processing is added, sharing system resources such as the buses, CPU, and memory, the overall jitter for every process in the system is significantly increased, which has a detrimental effect on maximum guaranteed response time and/or determinism.

Therefore if dedicated hardware that can off-load the IP protocols are available on the system, they must be used. And if not available, then it shall be considered to include them in the final hardware design. A great care must also be taken in the choice of the IP stack which performance must be measured prior to add any other protocol on top of it.

Compression mechanisms like MJPEG, often used to reduce the amount of data to transfer, must be avoided as they introduce an unknown latency. For some applications like the medical ones, it is even forbidden to compress images as it might introduce artifacts or remove details from the original picture.

B.30.5 Instantiation of the recommended implementation method in the reference platform

Implement different pipelines using Gstreamer or any low-latency protocols (e.g. one for the display and another from the sensor). To keep benefit from a low-latency streaming protocol, it will be necessary to perform image processing using zero-copy libraries such as OpenVX.

B.30.6 Evaluation of the guideline in reference applications

This guideline is not actually used in the medical use case of the project while it is using the SD-card as the input and the HDMI as the output, but it is used for the final product receiving images from express PCI and sending the processed images over GigE Vision.

B.31 Major challengers to integrating hard real time image processing using neural networks in MPSoCs

B.31.1 Guideline Information

Guideline Responsible	Alvin Sashala Naik, Thales
Guideline Reviewer	Pedro Machado, Sundance
Guideline Audience	Application developers
Guideline Expertise	Application developers, System Architects
Guideline Keywords	Neural Networks, Computer vision, Deep-learning

B.31.2 Guideline advice

To integrate Artificial Neural Networks in low-power embedded computer vision in embedded and real-time constraints it is necessary to rework the network and the methodology to use is very similar to other applications but at a higher level of abstraction.

B.31.3 Insights that led to the guideline

There are barriers to implement computationally intensive deep learning algorithms in low-power embedded system-on-chips considering real-time constraints. Because the embedded resources are limited in terms of computing power and in terms of memory bandwidth & size. It is very often impossible to execute a neural network designed on servers directly on an embedded target. The model developed after training of a neural network do not consider any constraints like memory sizes.

Some processors, designed for neural network execution or already well adapted to accelerate the computation taking place in neural networks, come with a tool chain that will adapt the operators of the neural network to their target.

When such a tool chain does not exists, or when this tool chain is not able to shrink the network on the embedded platform, some steps have to be done manually.

B.31.4 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)
3. modify the basic operators of each layers

4. Because of the kind of computation used during the learning methods, the networks are trained with floating points. A conversion to integer will reduce the number of computation and allow to target simpler chips and FPGAs.
5. Reduce the dynamic of the integer data. It has been shown that 16 bits integer can be used with the same performance as with the floating points and 8 bits only has a marginal effect on the network accuracy. Some layers, like the last layers doing the classification can even be binary. Low-power oriented chips implement 4-bit, 8-bit and 16-bit operators to deal with such optimized networks.

This goes farther than optimizing the source code of the application and require to change the algorithm itself. Some of the changes, like modification of the dynamic requires to redo a part of the learning as it will modify the weights of the neurons.

B.31.5 Instantiation of the recommended implementation method in the reference platform

This has impact on the selection of the target. An FPGA is a well suitable device that is very well adapted to deal with several dynamic and can highly benefit from binary data.

Because the Tulipp platform goal is to develop an image processing platform and because neural networks are now the state-of-the-art algorithms to analyze images, the platform is ready for these AI applications.

B.31.6 Evaluation of the guideline in reference applications

Neural networks were not implemented in the applications during the project, but a PhD work is currently starting and will use this platform for neural network development. A following project will also be proposed to add neural network analyze in robots based on the Tulipp platform.

B.32 Ensure Dataflow for FPGAs by using HLS Inline Subfunctions

B.32.1 Guideline Information

Guideline Responsible	Boitumelo Ruf, Fraunhofer
Guideline Reviewer	Philippe Millet, Thales
Guideline Audience	Toolchain designers, Application developers
Guideline Expertise	Application developers
Guideline Keywords	Code Optimisation, FPGA

B.32.2 Guideline advice

FPGAs have a limited amount of memory and are therefore designed to process data in a streamed manner. Especially in the development of smaller subfunctions, data streaming can become quite cumbersome. A simple way to ensure data flow is to declare these subfunctions as `inline`.

B.32.3 Insights that led to the guideline

When trying to export code into subfunctions, the data flow can be interrupted. This might lead to inefficient code generated by HLS not meeting the preferred initiation interval needed in order to pipeline the operations. Which can ultimately result in a failure to meet the timing constraints of the platform.

Streaming the data between every function often comes with additional work. Inlining subfunctions proposes a simple alternative for reducing the complexity of the code, making it more comprehensible while ensuring data flow.

B.32.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

If streaming data to a subfunction 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.

In order to find out whether a function needs to be inlined or whether the inlining process was successful, a performance estimation with respect to timing measurements and data flow can be helpful. Such an analysis is typically part of the HLS tool provided by the vendor (e.g. Xilinx SDx).

For more information about inlining subfunctions see the SDx Pragma Reference Guide [1].

B.32.5 Instantiation of the recommended implementation method in the reference platform

Implementation examples are provided in the referenced guide.

B.32.6 Evaluation of the guideline in reference applications

While Implementing the semi-global-matching algorithm for the UAV use case, inlining subfunctions has been a solid method for exporting code into subroutines without hurting the pipeline. They can be used to structure the code without loss in performance.

B.32.7 References

[1] SDx Pragma Reference Guide, `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1253-sdx-pragma-reference.pdf`

[2] Vivado HLS User Guide, `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf`

B.33 Avoid using heavy libraries while writing source code for embedded systems

B.33.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Najdet Charaf, TUD
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	Libraries, hardware optimisation, cross-compilation

B.33.2 Guideline advice

Avoid linking against heavy libraries (e.g. LibTIFF, openCV...) while porting the application onto embedded systems. Even though such libraries will save time during the design of the algorithm, taking care of the image manipulation and storage, it will also strongly influence the design of your code and use a lot of memory resources.

B.33.3 Insights that led to the guideline

While we were porting the medical use case on the Tegra family processors, we got difficulties to integrate LibTIFF library. One difficulty leads on the memory resources used by the library while doing image compression and decompression, another difficulty was the time spent by the library in such image manipulation. Such behavior does not match well with embedded systems. Another problem comes also while linking the application code against these libraries because they are usually not available on RTOS or bare metal. Then you have two options: 1. do not use the library and rewrite the source code without it 2. compile the library on the target

Option 2 is actually not an option as you will notice that the library has a lot of dependencies down to the operating system which makes it almost impossible to port to the embedded system.

B.33.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Write the source code based on light data formats (i.e. no compression) that can be manipulated with simple code. During development phase, also convert manually (or with scripts) all test images from complex format (e.g. TIFF) into either PPM/PGM images or hexdump format to avoid dependency issues during cross-compilation for RTOS implementations.

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.

B.33.5 Instantiation of the recommended implementation method in the reference platform

The reference platform is opened, all the specifications are available, and do not require any specific data format. The application developer has to take care of the data format and data conversion. Since we deal with embedded systems, we did not include any libraries from the Linux world. Because the platform includes an FPGA, it is also highly recommended to stream the data in the FPGA rather than letting the FPGA access the data from an external memory bank.

B.33.6 Evaluation of the guideline in reference applications

For the medical use case we removed all the dependencies to external libraries, changed the application to use only raw data formats and rewrote all the filters which were relying on these libraries. This allowed us to compile the source code onto the three targets we wanted to evaluate (Tegra X2, Myriad 2, Zynq). From this source code, we could then rework the source code to optimise it on each target.

B.33.7 Related guidelines

- B.28 Avoid LibTIFF library and use raw image format

B.34 Start from a pure C version of the application code before to optimize it for your embedded system

B.34.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Boitumelo Ruf, FHG
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	optimisation

B.34.2 Guideline advice

When porting an application to an embedded system, start by writing a pure C version of your application prior to do any optimisation of the code.

B.34.3 Insights that led to the guideline

When we worked on the medical use case, we got a C++ source code implemented on Linux and linked against several libraries. The first target we wanted to evaluate was the NVidia Tegra X2 SoC (TX2). Since the TX2 runs an Ubuntu Linux, we just recompiled the code with the libraries and let it run on the ARM processor. It was running but the performances were very poor and we wanted to make use of the GPU matrix. Because the code was based on image format and C++ classes that manipulated the images, it was tricky to identify the part of the code that we could change, but even when it was possible, we did not get the performances we were expected.

B.34.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Because optimisation will add target dependency on the source code, it is necessary that the code you start from has low to no dependency on code that you cannot modify. We advice that you get rid of the libraries that you cannot modify or that are too complex, and we advice also to use C code rather than C++ as it will also make it simpler for optimisation and maintainability of the source code.

When the application has to be ported on an operating system (OS) it might also not allow you to use C++ and, because each operating system has its own APIs, you will have to rewrite the parts of the code that have to rely on the OS libraries. These libraries might not be compatible with the libraries used on the PC that was used at first to develop the application.

B.34.5 Instantiation of the recommended implementation method in the reference platform

The reference platform implemented in Tulipp (hardware platform + operating system + libraries + tool chain) does not use C++ and relies on C code and the APIs of the HIPPEROS operating system. Since the code could be adapted to several platforms, we isolated the target dependent code in a set of files that has to be adapted when porting the software onto another hardware platform. We expect the application to have the same implementation strategy.

B.34.6 Evaluation of the guideline in reference applications

We rewrote the medical use case from C++ to C code, removing at the same time the library dependencies. From this pure C code we wrote an optimised version for Tegra X2 which allowed us to get the expected performances. Then the very same unoptimised C code was optimised again to a Myriad2 dependent version and to a Zynq dependant version. We therefore have five versions of the same application: 1. the C++ reference application developed on a Linux PC and based on Linux libraries 2. the pure C reference developed on a Linux PC which allowed us to compare the results with the C++ reference application 3. the Tegra X2 optimised version 4. the Myriad 2 optimised version 5. the Zynq optimised version. This version is using the whole Tulipp platform with the HIPPEROS and SDSoC is used to accelerate the processing on the FPGA matrix.

B.34.7 Related guidelines

- B.28 Avoid LibTIFF library and use raw image format
- B.33 Avoid using heavy libraries while writing source code for embedded systems

B.35 Make sure all the developers are using the same tools versions

B.35.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Antonio Paolillo, Hipperos
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	reduction of development risks

B.35.2 Guideline advice

All members of the development team shall use the same tools and hardware versions. For each tool, the version must be selected depending on all the constraints of all members. For the HW, on a given application e.g. the same version of FPGA must be used (for instance zynq 7000, zu3 or zu4), else several versions of hardware has to be taken into account for the development, which will be difficult to maintain over time.

B.35.3 Insights that led to the guideline

When we ported the medical application on the Zynq platform, we used SDSoC version 17.4. The main reason for that choice is that the ".4" version of the Xilinx tools is always more stable than the earlier. When we wanted to integrate the function we accelerated with SDSoC, on the Tulipp platform with the tool chain and the operating system we could not compile it because the SDSoC version of the Tulipp platform was 17.2 and the API were different so SDSoC were complaining.

The result of the discussions we had in the consortium was to port the Tulipp platform to use version 17.4 of the Xilinx tools.

B.35.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

When the components of the platform are selected and a decision is made on the tools that will be used in a development team, a decision must also be made on the tools versions to be used. Without this decision the integration might not be possible and there is a high probability that it will impact several parts of the project leading to delays.

B.35.5 Instantiation of the recommended implementation method in the reference platform

The reference platform is now using a set of tools for which the version is referenced. For SDSoC we selected 17.4.

For the hardware, we selected zu4 but since it was not available at the time we needed them, we started with zynq 7000 and zu3 and we got difficulties when the zu4 became available to have the operating system fully operational on this new platform. Which show how critical it is to select a configuration of the whole development environment and to share that configuration with all members of the team.

B.35.6 Evaluation of the guideline in reference applications

Since the medical application was developed with 17.4 and because the Tulipp platform is now using it, we have been able to implement the whole medical use case on the Zynq 7000 platform, a porting is foreseen on the zu4 after the end of the Tulipp project.

B.36 Implement your own optimised libraries but rely on APIs from existing libraries

B.36.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Asbjørn Djupdal, NTNU
Guideline Audience	OS designers
Guideline Expertise	OS designers, Application developers
Guideline Keywords	optimisation

B.36.2 Guideline advice

Implement your own optimised version of a subset of the libraries you need for your application while respecting the APIs of the original library.

B.36.3 Insights that led to the guideline

The use of libraries to develop application code allows to develop faster and safer as you don't have to rewrite always the same functions again and again for each new application. However, to benefit from the hardware accelerators and get the best performances out of the computing platform, it is required to optimise the code of the library to use the available hardware accelerators and to schedule the functions in order to maximize the utilisation of the hardware resources.

Even though some libraries, like OpenCV, are available on several targets (e.g. Intel processors or NVIDIA Graphical processors) they have not been optimised for all the set of architectures you might find. When the optimised code is available, it is often not optimised for all the family of processors. For instance, an given optimisation on the data size and array organisation for an NVIDIA Tesla GPGPU will most likely not be as efficient on a Tegra X2 GPGPU and will require another set of parameters for the data size and organisation.

For a given application domain, you may also have a set of filters that are combined in an efficient way. To this end, one will probably create a dedicated filter function for that particular filters chain which can be optimised all together to avoid calling several functions and thus improve the locality of the calculation and the performance of the application.

For all these reasons, it will be better to build a in-house library. But pay attention not to reinvent the wheel each time you create a new library.

Common libraries are often open source and the API is largely available and documented. It is always worth to understand how the API and the structure of the library was build and the underlying reasons that pushed people to produce that API. Some of the tricks that one may wonder why they have bin made the way it is become obvious when one tries to actually propose such an API and implement the library.

B.36.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Because a consistent, useful and documented API is not obvious to produce, it is better to understand the existing ones, to select a subset that is actually needed for the application domain and to implement the optimised version of this subset in the product. It will save a lot of time, bugs and will allow optimisation. At the same time, it will limit the size of the library to the minimal size actually needed by the application.

B.36.5 Instantiation of the recommended implementation method in the reference platform

During the Tulipp development of the operating system we also discussed the best way to implement libraries on top of the operating system so that the libraries can be optimised for both the hardware and the operating system. Since the application developers already had in mind a set of library they are used to, we selected a subset of these libraries - i.e. the main functions mostly used - and we chose to implement them while keeping the API.

B.36.6 Evaluation of the guideline in reference applications

A subpart of OpenCV has been developed on top of Hipperos. A demo application has been developed by Hipperos. Even though the use case application do not use OpenCV, it is actively used by the ecosystem developed around the project. After the tutorial we gave them during the Hipeac conference, they will start developing their application on top of our software stack.

B.37 Arrange your data according to what works best for the hardware architecture

B.37.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Muhammad Ali, TUD
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	optimisation

B.37.2 Guideline advice

While optimizing the application for a given target, the data structure and organisation in memory must also be optimised for the hardware accelerators to deliver their full potential.

B.37.3 Insights that led to the guideline

When we implemented the Medical use case on several targets, we had to arrange the data in a way that is more efficient for each target.

Developers that are used to general purpose processor don't care about the organisation of the data. They usually develop the application for best maintainability and reusability. But when an application code has to be ported on a processing unit with accelerators, then data organisation will impact drastically the performance.

B.37.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

GPGPUs are parallel processors which have been designed to efficiently use DDR bandwidth. There is a specific hardware part in the chip that deals with memory access and tries to organise the data to maximize the burst utilisation and to minimize the memory page swapping. GPGPUs also have internal memory hierarchy which is partly available to programmers. Combining the behaviour and the memory sizes make the GPGPU a complex processor for which it is impossible to predict the best way to organise the data arrays and to map it to the processors through the OpenCL or CUDA paradigms. So the best way is to try several configuration and to use the one that gives to the best performance.

FPGA are also parallel architectures but optimized for data streams. They are very good at processing flows of data with pipelined set of functions. According to this, it will not be efficient to let the FPGA access DDR memory randomly. The best way will be to implement the application the way it uses streams of data. FPGAs are particularly an a target of choice when you get data directly streamed out from the sensor. This can allow very low

latency computation and avoid to implement huge amount of memory which can save power consumption.

Myriad 2 processor is a processor designed for IoT and image processing. This is an SoC with 12 accelerators for image processing. These accelerators, called SHAVE, have access to a high-speed, local and dedicated memory called CMX. In order to get the best performance from the SHAVE it is necessary to load the data in the CMX memory. However, the CMX memory has 16 banks (1 bank per SHAVE) and is only 2MB which makes 128 KB per bank. The SoC is connected to a 128MB DDR (another version of the chip comes with 256 MB DDR) which allows to store at least a complete 1024x1024x4 Bytes image. The difficulty is then to schedule the data movements between the DDR and the CMX so that the processing never stops and to maximize the chip utilization. In order to do this, the data has to be organized in patches and the application is modified to work on windows sliding over the image.

B.37.5 Instantiation of the recommended implementation method in the reference platform

The libraries that come with the reference platform have been optimised to take memory organisation into account. Through the tool chain, the performance of a chosen data organisation can be evaluated for performance both in time spent for the processing and in required power consumption.

B.37.6 Evaluation of the guideline in reference applications

During the evaluation of several chips for the medical use case, the application was optimised for each target and the data was arranged to use the chips at its best performance potential. This allowed us to fairly compare the targets.

B.37.7 References

Myriad 2, https://www.hotchips.org/wp-content/uploads/hc_archives/hc26/hc26-12-day2-epub/hc26.12-6-hp-asics-epub/hc26.12.620-myriad2-eye-moloney-movidius-provided.pdf
Optimizing embedded software for power efficiency: Part 3 – Optimizing data flow and memory, https://www.embedded.com/design/power-optimization/4430266/optimizing-embedded-software-for-power-efficiency--part-3---optimizing-data-flow-and-memory-

B.38 Do not use dynamic memory allocation

B.38.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Habib ul hasan Khan, TUD
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	application development optimization

B.38.2 Guideline advice

Do not use dynamic memory allocation unless you actually need it.

B.38.3 Insights that led to the guideline

Dynamic memory allocation means that the amount of memory used by your application varies at run time.

Using dynamic allocation has several drawbacks:

1. it is time consuming
2. it creates holes in memory
3. it can fail in finding the required memory amount

Therefore, the data must be statically allocated. This allows the developer to know at compile-time if the application memory needs can be fulfilled by the hardware platform and does not require time at run-time to find a proper memory location for the data.

B.38.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

When programming in C, this is what we get when calling the malloc() and free() functions. It is also used in C++ while creating or destroying objects.

Malloc() is used by the application to tell the operating system that a given amount of memory is necessary for the application. When calling the malloc() function, the operating system will scan the memory to find a hole (an empty region) where the requested data size can fit. When calling the free() function, that memory space will be marked as free again and can be selected again when calling the next malloc().

Scanning the memory is time consuming and it is not possible to predict how long it will take for the operating system to find an empty space for the requested data size. Another problem comes after calling malloc() and free() with variable data size holes appears in memory. After

several such malloc()/free() sequence the holes become too small to fit any data size and it is not possible to allocate any memory anymore. To overcome this problem, operating systems often implement a “garbage collector” which will compact the memory and remove the holes. The garbage collector is also a time-consuming task which must not be allowed with real-time application.

Embedded systems have much smaller memory that desktop PCs which increases the need to compact the memory.

Another dangerous aspect of the malloc() function is that it returns NULL when it cannot allocate memory. Malloc() results must be checked prior to be used as it will lead to segmentation faults if you try to use a pointer to NULL.

This is also the reason why one should avoid using C++ code on embedded real-time applications.

If dynamic allocation is really necessary, then it must be called at an initialization phase of the application and free() must never be used at run-time, i.e. after initialization.

B.38.5 Instantiation of the recommended implementation method in the reference platform

No dynamic memory allocation is used in the operating system, the libraries and the application at run-time.

B.38.6 Evaluation of the guideline in reference applications

The initial code of the medical use case was written on C++. Prior to port it to the embedded targets, we translated the code into C.

The C++ code could compile on the Tegra X2 tool chain, but the performances were very poor. While we translated the code into C, we also made the allocations static and rewrote the filters in C to deal with the static allocated data.

B.39 Optimisation method

B.39.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Boitumelo Ruf, FHG
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	Code optimisation strategy

B.39.2 Guideline advice

Define and use metrics before any optimisation and focus on the most impacting parts of your application first. Be ready to also define error margins for your reference output data.

B.39.3 Insights that led to the guideline

When you port an application on a target and optimise it to get higher performance you often face a dilemma: “what part of the code to start with?”.

Since the first implementation of the application if often developed on a desktop PC and written with high abstraction level languages, it is difficult to figure out what part of this code is the most demanding for resources (memory, CPU time. . .).

Optimisation is a risky and time consuming activity. The main risk is to produce a code that will not produce the very same results as the reference one. Changing the operations to optimise a computation might change the computation noise and change the final values of the data.

Since the optimisation process will require time, starting optimising the source code without any clue on metrics, performance indicators you want to optimise and their initial values is very like finding a way with no GPS or map.

B.39.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

The first recommendation is to define margins on the output reference test data. Too often, the processing is validated by a bit to bit comparison of the output results while the really important fact is to still get a meaningful result. This allows the developers to modify the application code and to check that they achieved their goal by getting correct meaningful results within the error margins.

The second recommendation is to define the performance indicators that are required for the implementation of the application. It could be the used memory resources, the CPU time spent in functions, the latency of a computation or the power-consumption of the whole

application. Whatever is meaningful for the application must be measured. The initial value of each indicator must be known and have to be monitored throughout the whole optimisation process. The initial value is firstly used to help the developers figure out the parts of the code that need to be optimised in priority because of the high proportion of their impact on the indicators versus the total impact of the application. Proceed function by function, starting from the most impacting one.

B.39.5 Instantiation of the recommended implementation method in the reference platform

For the reference platform we developed means from the tool chain, through the operating system and down to the hardware to be able to get measurement from the application. The platform allows the application developer to put tags in the code that will allow an identification of the part of the application that has to be measured. The consortium also developed a board to measure the power consumption. The measurements from this board can be synchronised with the tags which allows an identification of the most power demanding parts of the application.

B.39.6 Evaluation of the guideline in reference applications

Each use case defined Key Performance Indicators (KPIs) at the very beginning of the project. The KPIs were measured on the first implementation of the application and are monitored during the development. For the medical use case we used 2 main KPIs: - the processing latency of an image: because surgeons need to have a short latency when they move their hands they must see what they do on the screen. - the power-consumption: because the final product is embedded in a closed cabinet and can only evacuate a limited part of the heat.

B.39.7 Related guidelines

- B.1 Adjust Your Algorithm to the Underlying Architecture

B.40 Manage interrupts carefully

B.40.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Magnus Peterson, Synective
Guideline Audience	Application developers
Guideline Expertise	OS designers, Application developers
Guideline Keywords	optimisation

B.40.2 Guideline advice

Interrupts must be managed with great care not to impact the performance of the system. The processor must not spend too much time in interrupts.

B.40.3 Insights that led to the guideline

Real-time applications typically receive data and process it at the same pace. Generally the sensor delivers an interrupt when data is available. While the interrupt allows the processing system real-time reaction to the incoming data, the management of this interrupt can lead to very poor performance if done with no care.

B.40.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

A very poor way to manage interrupts would be to call the processing function from the interrupt itself, because it then prevents the system from being able to manage any new interrupts. The full processing must be done outside of the interrupt routine. The handling of the interrupt must be as short as possible, taking care of for example buffer management by updating its status.

If the system spends too much time inside an interrupt and cannot serve all the pending interrupts that occurs while already serving a previous interrupt, then the system might lose interrupts and data can also be lost. Thus the system will not be able to keep processing data at real-time. The system will also not be able to answer other requests which might lead the other components on the system to wait for a response and slow down the whole processing chain.

Do not forget that the processor will need to save registers or swap context while entering an interrupt. So, another impact of having too much code in the interrupt is the necessity to save a lot of registers, which takes a long time, each time the processor enters anew the interrupt management code.

The programmers generally allocate a circular buffer in the global memory to receive the sensor data. The depth of this circular buffers must be computed according to the data flow to overcome potential concertina effect and keep the ability of the system to receive the sensor data.

Accelerators often use long processing pipeline. This is very efficient to process big bursts of data. Since interrupts will break the instruction pipeline, interrupts must be avoided or reduced in these computation phases. A good way to manage this is to prevent the system from entering into interrupts by masking the interrupts and then unmasking them periodically after each burst of data is computed. The size of the burst is to be defined by the application developer and the system architect.

B.40.5 Instantiation of the recommended implementation method in the reference platform

The management of interrupts is done through the operating system.

B.40.6 Evaluation of the guideline in reference applications

In the reference application, we use interrupts to manage the incoming data from the camera. The data is received in a buffer in DDR. The camera sends an interrupt which is received by the operating system. The operating system manages the buffers and sends a signal to the application. The application is informed that data is available and starts to process it asynchronously.

B.41 Avoid task swapping

B.41.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Björn Gottschall, NTNU
Guideline Audience	OS designers, Application developers
Guideline Expertise	OS designers, Application developers
Guideline Keywords	optimisation

B.41.2 Guideline advice

If you need to execute several tasks in parallel, you have to control their scheduling to avoid to spend too much CPU time in the swapping mechanism.

B.41.3 Insights that led to the guideline

Multi-tasking is the ability for a platform (hardware components and operating system) to deal with several concurrent tasks in parallel. Even though your platform can handle multi-tasking, running several tasks or applications in parallel, the actual fact is that the tasks are concurrent and using the same resources.

On a mono-core processor, the tasks are actually executed interlaced, one at a time sharing the processor-time. Changing from one task to another is called task-swapping. It is done by the operating system that schedules the tasks execution on the processor.

Swapping from one task to another requires the operating system to save the context of the current task in order to idle it and restore the context of the next task in schedule to be able to continue its execution. The saving / restore mechanism requires many memory-copy operations and is time consuming. When too many tasks have to be scheduled and / or when the tasks are swapped too often, then a tremendous time is spent in the swapping operation and lost for effective computation.

On multi-core processor, the operating system usually hides the cores to the users and manages the mapping and scheduling of tasks. It is however usually possible to configure these operating systems to manually attach each process (or task) on a defined core to avoid task-swapping.

On many-core processors, the operating system often see the cores as separate mono-core processors and allow the users to map their tasks on each core.

B.41.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Write the application to control the tasks yourself or chose operating systems with scheduling policies that will avoid task swapping or that allows you to fine-tune the scheduling of your tasks.

While task-swapping already causes a performance issue while it takes CPU time for internal scheduling, it is a hazardous for real-time application as tasks will fail to meet the deadlines because their execution time budget will be reduced. Therefore, real-time application should always have a strong scheduling mechanism and the users should be able to configure the scheduling policy.

B.41.5 Instantiation of the recommended implementation method in the reference platform

The Maestro operating system of Hipperos developed in Tulipp allows the user to define his own scheduling policies. The tool chain developed by NTNU helps the user to define the right mapping and scheduling for the application to run efficiently.

B.41.6 Evaluation of the guideline in reference applications

Such a mechanism allows tasks to be segregated over time and never execute at the same time on the same processor. This removes the need for the operating system to swap the tasks. The swapping being defined at design time.

On the medical use case the application is hard real-time continuously taking X-ray images to produce a video stream that allows the surgeon to see inside the body of a patient during surgery. The picture is taken by sending radiation through the body of a patient. Regulation law says that you cannot send radiation to a human if it is not for a purpose. The purpose of the device is to take X-ray images. So each photo that went through the patient must be used to deliver a picture for the medical staff. Therefore, it is not allowed to lose any frame.

This constraint have been given to the operating system to schedule the tasks. The operating system allows the user to define the tasks periodicity and their deadlines. For the medical application we defined a the periodicity equal to the frame rate and the deadline equal to the maximum allowed latency of 70ms. With this configuration we know the system are always deliver the result on time. If, for some reason, the task would last more than the given 70ms, then we would be informed by the operating system and we could investigate that particular case, which most likely would be resulting from a bug in the task itself.

B.42 Use TDMA to increase stability and predictability

B.42.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Magnus Peterson, Synective
Guideline Audience	Application developers
Guideline Expertise	Application developers
Guideline Keywords	Real-time execution, critical applications

B.42.2 Guideline advice

For these applications (real-time and critical), use segregation of the tasks over time to share the resources.

B.42.3 Insights that led to the guideline

Critical applications require to ensure that the execution time of a given task will never be compromise whatever event arise and whatever other task is concurrently executed on the processing unit.

Real-time application require almost the same capabilities as the execution of the tasks must be as stable as possible, limiting the execution time jitter as much as possible to avoid a wrong scheduling of the tasks that will lead to the necessity, in image processing, to drop some frames in order to come back to a stable situation.

B.42.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

TDMA stands for Time Division Multiple Access. It schedules access from software tasks to a hardware resource during slots of time. The utilization time of a hardware resource is thus divided into slots. Each task is given a set of slots through a sequence that repeats periodically.

The time slots allows time isolation between tasks. Each task can get the feeling it is executing alone on the system while when it executes, there is no other concurrent task running.

This way of building system simplifies the integration while the system is then turned into a composable architecture. Since each task is isolated from the others, the properties of that task (in particular execution time) will not change when other tasks are added and integrated.

This way of scheduling is largely used in real-time operating systems as it allows the implementation of hard real-time constraints. The task is only allowed to execute on its time-slots and is preempted when its time is out.

One drawback of this method is the potential to lose CPU time with empty slots. This comes when a task does not have anything to process during its time-slot. Since the time-slot allocation is static (for real-time reasons) and defined at design-time, there is no possibility to use that empty slot to execute another task.

B.42.5 Instantiation of the recommended implementation method in the reference platform

The operating system of the Tulipp project allows TDMA scheduling.

B.42.6 Evaluation of the guideline in reference applications

It is already used in demo applications that come with the operating system of Hipperos.

This methods is to be implemented on the medical use case while since pictures are captures using x-ray radiations we have to certify that we never loose any frame, since each photons that went through the patient to produce an image must be served to the doctors.

It must also be used in the UAV use case as the drone might crash if it drops too many frames.

B.43 Fine tune the send and receive buffers

B.43.1 Guideline Information

Guideline Responsible	Philippe Millet, Thales
Guideline Reviewer	Boitumelo Ruf, FHG
Guideline Audience	Application developers, System architects
Guideline Expertise	Application developers, System architects
Guideline Keywords	optimising the communication

B.43.2 Guideline advice

The send and receive buffers of the communication mediums must be sized for each application big enough to avoid the concertina effect but as small as possible to use as little memory resources as possible.

B.43.3 Insights that led to the guideline

Concertina effects can occur when a data flow or a pipeline has to stop at some stage. Like in traffic jam, it will then propagate in the pipeline and reduce the performance of the whole system.

This effect arise mostly when a task in the pipeline has variability (either it needs a variable time to compute or it delivers a variable amount of data). This variability has to propagate in the system and will impact the neighbor tasks. To limit the impact of variability, input and output buffers are used to isolate one task from the behavior of the neighbors. The upstream neighbors will write their output buffers. A communication medium (most likely with a DMA) will transfer the data from that output buffer to the downstream neighbors' input buffers.

B.43.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

The basic implementation is to have a total of 4 buffers. 2 input buffers and 2 output buffers: - One input buffer is used for the task to read data from, the other input buffer is used by the communication medium to write data in. - One output buffer is used by the task to write the processed data, the other output buffer is used by the communication medium to read data from.

Having these buffers allows the processor to have two buffer reserved for its computation. It can freely work on the input data without locking the DMA that would have to wait for the processing to be finished before loading data again in the input buffer. This allows a simultaneous work of both the CPU and the DMA as they work on their own buffers. When the CPU has finished processing the data from the first input buffers, then it start processing

the data from the second input buffer, while the DMA will load new data in the first input buffer.

But in a pipeline of tasks, the execution of each task might be different from the execution of the other tasks. And their execution time might also vary over time. Therefore it is necessary to adapt the number of buffers to a number that allows the whole application to overcome the concertina effect.

Since the required number of buffers strongly depends on the scheduling of the application, it must be given the application developer a mean to configure the number and the size of these buffers.

B.43.5 Instantiation of the recommended implementation method in the reference platform

The operating system allows the user to configure the communication medium and to adjust the number of buffers. One way to allow the user to define the number of buffers that are necessary for his application is to provide the user with an API to allocate the buffers. Another method is to let the user allocate memory and only ask the user to provide the communication medium with the proper pointers to these buffers.

In Tulipp we selected the second way.

B.43.6 Evaluation of the guideline in reference applications

In the medical application we actually use 2 input buffers and 2 output buffers.

B.44 Use Decouplers on the Input and Output AXI Stream ports for Partial Reconfiguration

B.44.1 Guideline Information

Guideline Responsible	Ariel Podlubne, TUD
Guideline Reviewer	Philippe Millet, THL
Guideline Audience	HW designers
Guideline Expertise	HW designers
Guideline Keywords	Partial Reconfiguration

B.44.2 Guideline advice

When an application requires the use of Partial Reconfiguration techniques, the static design has to be modified accordingly. The interfaces of the reconfigurable module must be decoupled while reconfiguring.

B.44.3 Insights that led to the guideline

During reconfiguration, the FPGA re-writes the designated partition for this task. Therefore, not expected data could be transferred from and to said partition which could affect later stages of the design with incorrect, corrupt or undesirable data.

B.44.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

An instance of a decoupler [1] should be added to all master and slave AXI-Stream interfaces for image processing applications. This will prevent unwanted data-flow being transferred in and out of the reconfigurable module during reconfiguration.

B.44.5 Instantiation of the recommended implementation method in the reference platform

SDSoC demo running multiple filters from the Image Processing Library.

B.44.6 Evaluation of the guideline in reference applications

A proof-of-concept has been implemented on the paper “Low Power Image Processing Applications on FPGAs Using Dynamic Voltage Scaling and Partial Reconfiguration” [2]

B.44.7 References

[1] Partial Reconfiguration Decoupler v1.0 LogiCORE IP Product Guide (PG227), https://www.xilinx.com/support/documentation/ip_documentation/pr_decoupler/v1_0/pg227-pr-decoupler.pdf

[2] Low Power Image Processing Applications on FPGAs Using Dynamic Voltage Scaling and Partial Reconfiguration, https://ieeexplore.ieee.org/document/8596910

B.45 Dynamic voltage and frequency scaling DVFS on ZC702

B.45.1 Guideline Information

Guideline Responsible	Muhammad Ali, TUD
Guideline Reviewer	Philippe Millet, THL
Guideline Audience	Application developers, System Architect
Guideline Expertise	HW designers
Guideline Keywords	Library, FPGA, DVFS

B.45.2 Guideline advice

Only the power of the programmable logic (PL) should be measured to improve computation rate.

Voltage scaling of the programmable logic (PL) should be done carefully, since any mistake can cause permanent damage to the board.

For dynamic frequency scaling a reconfigurable clocking wizard can be added for the PL design. The clock input for PL data movers generated by SDSoC should be provided from the PS and it should not be scaled.

B.45.3 Insights that led to the guideline

Some key features of using the DVFS system on the ZC702 board are mentioned, which should be treated carefully while using this application. DVFS was implemented on image processing functions and is discussed in detail in D3.2.

B.45.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

The application used for monitoring the power consumption of each power rail on ZC702 is from [1]. This application measures all the power rails of the board which results in a very slow processing speed. It was modified and optimized to measure only the power of the Programmable Logic (PL) for this project to improve computation rate.

Dynamic voltage scaling of PL should be set carefully, since it may cause permanent damage to the board if the selected voltage exceeds the limits described in the manual [2]. Power management of processing system (PS) is not advised, since the monitoring application will consume some power and will not provide accurate results [3].

Dynamic frequency scaling was done using a reconfigurable clock on the PL. The frequency of the clock was set by writing to the configuration registers of the clocking wizard. Configuration registers should be set according to a formula mentioned in the manual [4].

For dynamic frequency scaling a custom platform in SDSoC was developed, so a reconfigurable clocking wizard can be added for the PL design. It is important that clock input for PL data movers generated by SDSoC should be provided from the PS and it should not be scaled. Only frequency scaling of the PL design is done using clocking wizard.

B.45.5 Instantiation of the recommended implementation method in the reference platform

The DVFS application is proposed for ZC702 FPGA board. Dynamic frequency scaling can be implemented on Tulipp board but for voltage scaling and power monitoring additional hardware (e.g. power rails present in ZC702) on the Tulipp board is required.

B.45.6 Evaluation of the guideline in reference applications

In this guideline, different variables (to implement DVFS) are discussed which needs to be set correctly as they may cause permanent damage to the board if not selected correctly.

B.45.7 References

[1] Xilinx, Zynq-7000 AP SoC Low Power Techniques part 3 - Measuring ZC702 Power with a Standalone Application Tech Tip, https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842356/Zynq-7000+AP+SoC+Low+Power+Techniques+part+3+-+Measuring+ZC702+Power+with+a+Standalone+Application+Tech+Tip
[2] Xilinx, Zynq-7000 SoC (Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015, and Z-7020): DC and AC Switching Characteristics, Xilinx, 2018, https://www.xilinx.com/support/documentation/data_sheets/ds187-XC7Z010-XC7Z020-Data-Sheet.pdf
[3] Accurate Power control and monitoring in ZYNQ boards by Arash Farhadi Beldachi and Jose L. Nunez-Yanez, https://ieeexplore.ieee.org/document/692741
[4] Xilinx, Clocking Wizard v6.0, 2018, https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v6_0/pg065-clk-wiz.pdf

B.46 Using Dynamic Partial Reconfiguration on a Xilinx device controlled by an Operating System on the Processing System

B.46.1 Guideline Information

Guideline Responsible	Julian Haase, TUD
Guideline Reviewer	Philippe Millet, THL
Guideline Audience	Application Developers, HW Designers
Guideline Expertise	HW/SW Designers
Guideline Keywords	Dynamic Partial Reconfiguration

B.46.2 Guideline advice

This guideline assumes that the operating system manages partial reconfiguration in the FPGA by providing the drivers as explained in section 6.2 of public deliverable D1.3.

General:
- At first, you should build a working stand-alone hardware solution which is tested without the operating system. - Additionally, it is recommended to use decoupler[21] in the hardware design. - After that you write your application including the code controlling the dynamic partial reconfiguration.

Specific: - Check, which reconfiguration interface (e.g. PCAP or ICAP in Xilinx devices) are supported by the operating system. - Use the method which utilizes the lesser hardware (in this case PCAP). Insights that led to the guideline
Usually, several bugs are expected during the development phase. If you are developing an application for an operating system which is using the programming logic of a device combining processors and FPGAs on one chip, then two big areas for possible errors arise. On one hand there is the software part and on the other hand there is the hardware part. You can minimize your application development time, if you validate the hardware with the dynamic partial reconfiguration as stand-alone before writing the code for your application.

B.46.3 Recommended implementation method of the guideline along with a solid motivation for the recommendation

Bitstream preparation: - Build a hardware design with reconfigurable modules. - Generate the static and the partial bitstreams. - Test the bitstreams as stand-alone application on the hardware.

General procedure:

- Load the static and partial bitstream into the local memory of the processor.
- Configure the programming logic the first time with the static bitstream.

[21] B.44 Use Decouplers on the Input and Output AXI Stream ports for Partial Reconfiguration

- Decouple the reconfigure module.
- Load the partial bitstream of the reconfigurable module via the reconfiguration interface into the programmable logic.
- Connect the reconfigure module to the rest of the hardware.

B.46.4 Instantiation of the recommended implementation method in the reference platform

Two operating systems were used: 1. HIPPEROS: the choice of the TULIPP consortium for the reference platform 2. FreeRTOS: portable and open-source operating system suited for embedded applications

For using Dynamic Partial Reconfiguration and SdSoC refer to public deliverable D4.4[22].

B.46.5 Evaluation of the guideline in reference applications

For both operating systems a proof-of-concept has been implemented and tested. The acceleration of some functions in hardware with Dynamic Partial Reconfiguration gives the advantage of a better resource utilization which means you could use smaller FPGAs or using more different accelerators.

The developer could face some issues because the functionality is restricted to the given implementation of the drives, e.g for now only AXI Lite, not AXI Stream, as interface between the Processing System (PS) and Programmable Logic (PL) is feasible.

B.46.6 References

[1] Xilinx Zynq-7000 SoC - UG585, `https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf`
[2] XilFPGA Library, `https://github.com/Xilinx/embeddedsw/blob/master/lib/sw_services/xilfpga/doc/xilfpga.pdf`

B.46.7 Related guidelines

- B.44 Use Decouplers on the Input and Output AXI Stream ports for Partial Reconfiguration

[22] `http://tulipp.eu/wp-content/uploads/2019/01/d44-final.pdf`

B.47 Dynamic Partial Reconfiguration for TULIPP ZynqMP UltraScale

B.47.1 Guideline Information

Guideline Responsible	Ahmed Kamal, TUD
Guideline Reviewer	Fatima Kishwar
Guideline Audience	HW Designers , Application developers
Guideline Expertise	HW Designers
Guideline Keywords	Dynamic Partial Reconfiguration ZynqMP UltraScale+

B.47.2 Guideline advice

Dynamic Partial Reconfiguration (DPR) is supported for the TULIPP ZynqMP UltraScale+ platform by using the ICAP to partially reconfigure the FPGA. PCAP is by default the primary configuration port for ZynqMP UltraScale+. Xilfpga library is the current software driver from Xilinx to control the configuration of the ZynqMP UltraScale+ platform from the Processing System (PS) side. Unfortunately, the current version of the Xilfpga library (v4.1) does not support partial bitstream loading up to the time of writing this guideline [1]. The ICAP is recommended to be used instead of PCAP to overcome this limitation as explained in D4.4[23] chapter 4.

B.47.3 Insights that led to the guideline

The DPR process has to be managed from the PS side for the ZynqMP Ultrascale+ TULIPP platform.

B.47.4 Recommended implementation method of the guideline along with a solid motivation for the recommendation

The ICAP is used as the configuration port to partially reconfigure the Programming Logic (PL). ICAP is accessed through the PS unit by inserting the Xilinx AXI HWICAP IP core [2] on the PL to communicate between the PS and the ICAP primitive on the PL. Therefore, the DPR process is controlled by the PS with the same behaviour as if the PCAP is used. The Xilfpga library is required to be included to the software libraries running on the ARM A53 to deselect the PCAP from the Configuration Security Unit (CSU) and select the ICAP as the configuration port by setting the (CSU_PCAP_CTRL) register to be 0x0 [3]. Also, the HWICAP driver is included to the software libraries to manage the partial bitstream loading from the DDR to the ICAP using the AXI HWICAP core through the PS.

[23] http://tulipp.eu/public-deliverables/d4-4-final-tool-chain/

B.47.5 Instantiation of the recommended implementation method in the reference platform

The developer has to add the Xilinx AXIHWICAP IP core to the system block design together with the other developed modules.

B.47.6 Evaluation of the guideline in reference applications

The evaluation has been done on the TULIPP Ultrascale+ board using a reconfigurable image processing application.

B.47.7 References

[1] Xilinx. Zynq Ultrascale+ MPSoC Software Developer Guide, Appendix L : XilFPGA Library (v4.1), `https://www.xilinx.com/support/documentation/user_guides/ug1137-zynq-ultrascale-mpsoc-swdev.pdf`
[2] Xilinx, AXI HWICAP, LogiCORE IP Product Guide, PG(134) v3.0, `https://www.xilinx.com/support/documentation/ip_documentation/axi_hwicap/v3_0/pg134-axi-hwicap.pdf`
[3] Xilinx, Zynq Ultrascale+ Device, Technical Reference Manual, UG(1085) v1.8., `https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf`

B.47.8 Related guidelines

- B.44 Use Decouplers on the Input and Output AXI Stream ports for Partial Reconfiguration
- B.46 Using Dynamic Partial Reconfiguration on a Xilinx device controlled by an Operating System on the Processing System