Supporting Utilities for Heterogeneous Embedded Image Processing Platforms (STHEM): An Overview

Ahmad Sadek[1], Ananya Muddukrishna[2], Lester Kalms[1], Asbjørn Djupdal[2], Ariel Podlubne[1], Antonio Paolillo[3], Diana Goehringer[1] and Magnus Jahre[2]

[1] TU Dresden, Germany
{ahmed.sadek,lester.kalms,ariel.podlubne,diana.goehringer}@tu-dresden.de
[2] Norwegian University of Science and Technology (NTNU), Norway
{ananya.muddukrishna,asbjorn.djupdal,magnus.jahre}@ntnu.no
[3] HIPPEROS S.A., Belgium
antonio.paolillo@hipperos.com

Abstract. The TULIPP project aims to simplify development of embedded vision applications with low-power and real-time requirements by providing a complete image processing system package called the *TULIPP Starter Kit*. To achieve this, the chosen high-performance embedded vision platform needs to be extended with performance analysis and power measurement features. The lack of such features plagues most embedded vision platforms in general and practitioners have adopted ad-hoc methods to circumvent the problem. In this paper, we describe four generic utilities that complement and refine the capabilities of existing platforms for embedded vision applications. Concretely, we describe a novel power measurement and analysis utility, a platform-optimized image processing library, a dynamic partial reconfiguration utility, and an utility providing support for using the real-time OS HIPPEROS within Xilinx SDSoC. Collectively, these utilities enable efficient development of image processing applications on the TULIPP hardware platform. In future work, we will evaluate the relative benefit of these utilities on key embedded image processing metrics such as frame rate and power consumption.

Keywords: Embedded vision, image processing, performance analysis, profiling, low-power, dynamic partial reconfiguration

1 Introduction

Image processing is an application domain that deals with image manipulation, transformation and analysis. Images are a diverse class of input data since their width, height and pixel-depth depends strongly on the sensor used to acquire it. The wide variety of sensors and application types makes image processing a

This is the author version of the work which appears in ARC 2018. The final publication is available at Springer.

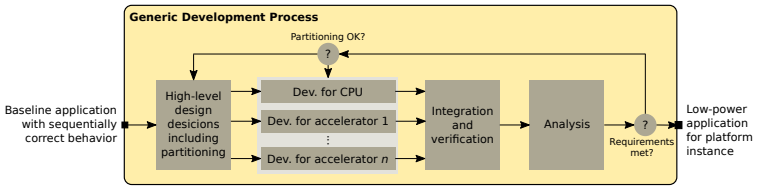

Fig. 1. The generic development process

complex and diverse application domain. Due to the large data volumes, high performance is needed to analyze images in systems with real-time constraints. Furthermore, image processing systems are often deployed in scenarios where power, energy, weight, cost and physical size are first-order constraints. The result is an overwhelming challenge for developers.

The overall objective of the *Towards Ubiquitous Low-power Image Processing Platforms (TULIPP)* project is to reduce the magnitude of this challenge by providing a complete image processing system package that developers can leverage towards their specific embedded application [9]. We refer to this package as the *TULIPP Starter Kit (TSK)* which consists of a a reference handbook, project applications and a platform instance. The reference handbook is a high-level best-practice introduction to embedded low-power image processing which is complemented by a collection of concrete, validated guidelines for embedded image processing system design. The project applications are industry-grade examples taken from the medical, automotive and unmanned aerial vehicle domains. Finally, the platform instance consists of a hardware platform, a real-time operating system and a collection of design and analysis tools. In this paper, we focus on the design and analysis tools that we have developed during the first year of the TULIPP project.

The main objective of the TULIPP tools is to contribute to substantially reducing the effort required to implement an image processing solution on selected heterogeneous platforms. The tools guide the developer through step-wise improvements to an image processing implementation and are designed to use the hardware technology and the operating system services developed in TULIPP. The overall development process is based on software optimization best practices by iteratively guiding the developer through successive changes to the code with the aim of achieving real-time performance with maximum energy efficiency. To maximize impact, we leverage existing tools where these are available.

We connect all components of the platform instance – hardware, RTOS, and tools – using an abstraction called the generic development process which is shown in Figure 1. The generic development process 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. The starting point of the generic development process is the baseline application that executes with correct sequential behaviour on a modern machine with a general-purpose processor. High-level partitioning deci-

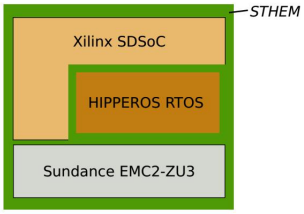

Fig. 2. The TULIPP-PI1 platform instance held together by STHEM to enable the generic development process [8]

sions decide which baseline functions should be accelerated and how. Partitioning splits off into accelerator-specific development stages that later join to produce an integrated application with the same correct behaviour as the baseline. The performance of the integrated application is checked against requirements. If found lacking, the partitioning and development stages are restarted. In this manner, programmers iteratively refine the baseline application to approach the required low-power and high performance features.

A platform instance can be created using any combination of hardware, RTOS, and development tools. However, support for the generic development process in each platform instance is unlikely to be readily available. For example, all the components of the TULIPP reference platform have independent workflows that partially overlap with the generic development process, and at a more basic level have poor to non-existent support for each other. We build utilities to resolve limitations of components of platform instances to ensure simplified support for the generic development process. Our utilities are collectively called *Supporting uTilities for Heterogeneous EMbedded image processing platforms (STHEM)*. STHEM is designed to be as vendor-independent as possible to simplify implementation for arbitrary platform instances. STHEM includes connecting glue that interfaces independent components together and standalone tools that extend individual components to provide complementary features.

The TULIPP toolchain is a combination of STHEM and existing components of a given platform instance that work together to simplify the generic development process for programmers. Figure 2 shows the TULIPP-PI1 platform instance which is platform instance that the TULIPP consortium is currently focusing most attention on. The reason for the attention is familiarity with the components that make up the platform instance. TULIPP-PI1 consists of the Sundance EMC2-ZU3 carrier board with the Xilinx Zynq UltraScale+ MPSoC processor arranged in a two-board configuration to expose a high degree of parallelism to applications. The hardware is operated seamlessly by the HIPPEROS RTOS. Application development tools in the platform instance are custom adaptations of Xilinx SDSoC and HIPPEROS tools to support multi-board acceleration and real-time requirements.

Table 1 lists the limitations of the main TULIPP-PI1 components and how our utilities alleviate these limitations. The current version of STHEM includes

Table 1. Limitations of TULIPP-PI1 components

Utility	EMC2-ZU3	HIPPEROS	SDSoC
Power Measurement Utility (PMU)	No power measurement hardware	Does not quantify task power consumption	Cannot correlate power consumption with application phases
HIPPEROS & SDSoC (HSCL)		Cannot accelerate tasks on FPGA	No support for HIPPEROS
HW/SW Image Processing Library (IPL)			Few optimized image processing functions

the three utilities that are necessary to provide a minimal end-to-end image processing system for the TULIPP-PI1 platform:

- The *Power Measurement Utility (PMU)* provides hardware support for measuring power in the EMC2-ZU3 and enables programmers to correlate instantaneous power samples with concurrent HIPPEROS application tasks and SDSoC's HW/SW traces.
- *HW/SW Image Processing Libraries (IPL)* enables high performance and productivity for commonly used image processing operations
- *Dynamic Partial Reconfiguration Utility (DPRU)* enables runtime reconfiguration of the FPGA fabric which can be used both within a single image processing algorithm and by the OS to switch accelerators at runtime.
- The *HIPPEROS SDSoC Compatibility Layer (HSCL)* adds HIPPEROS support to SDSoC, enabling programmers to accelerate HIPPEROS application tasks on FPGA accelerators

The rest of the paper is organized as follows. Section 2 describes the implementation of the PMU, IPL, DPRU and HSCL utilities which is the main contribution of the paper. We conclude the paper and indicate further work in Section 3.

2 STHEM Utilities

The STHEM utilities are a set of components that facilitate the development of low power image processing systems, shown in Figure 2. In the current phase of the project, they integrate different tools and components in a single suite to make them easier to use for developers.

2.1 Power Measurement Utility (PMU)

Improving the power efficiency of embedded applications begins with prudent device selection. For example, choosing FPGAs made with latest FinFET technology [1]. Once the device is fixed, power efficiency is refined to desired levels

in successive stages of profiling and optimization. Profiling uses power models during early design phases, and shifts to real-hardware measurements post-implementation. While standard power profiling methods are available for HPC-like systems [16], power profiling in embedded systems remains largely ad-hoc [17, 26, 2, 4, 13].

The Xilinx Zynq-based embedded platform that we have chosen for the applications of the TULIPP project, has poor support for power profiling. Neither hardware nor vendor tools have support for measuring power consumption at runtime. This complicates selection of application phases to direct power optimizations and makes it difficult to judge whether low-power requirements are met. Ultimately, a key contribution of the project – design guidelines for low-power embedded vision – cannot be demonstrated.

To solve this problem, we first looked towards solutions recommended by vendors. Xilinx recommends adding on-board current sensors such as precision shunt resistors to provide current measurements to the XADC [27], a hard-IP block in the FPGA substrate of the Zynq. A better solution, also recommended by Xilinx, is to replace the voltage regulators on the hardware platform with digital power controllers from Texas Instruments (TI) to measure current and voltages supplied to all power planes [24]. Another option is to use special measurement-friendly variants of the embedded platform built by third-parties [23]. While useful, these recommended hardware modifications were prohibitive due to cost reasons. We also deliberated about a model-only approach, i.e., use the Xilinx-provided power model called the *XPE* [26] to refine power efficiency as much as possible during early design phases. However, XPE can at best provide coarse-grained estimates and cannot correlate power problems with application phases.

We decided in the end to build external, cost-effective measurement hardware, complemented by specialized profiling software, to diagnose power problems of TULIPP applications at runtime and, in general, advance the state-of-the-art in power profiling of embedded vision applications.

Power profiling implementation: Our power profiling approach is packaged as the PMU. It essentially consists of an external measurement board that communicates power measurements to profiling tools on the host computer, as shown in Figure 3. The external measurement board is custom-designed and has multiple current sensors that measure power consumed by individual *units of interest* on the embedded platform, i.e., the EMC2-DP board in TULIPP-PI1. Profiling tools collect additional profiling data from EMC2-DP and analyze it together with power measurements to diagnose problems. Problems are shown on various visualization widgets, some of which are part of existing vendor tools.

We developed an external measurement board which we call *Lynsyn*. Lynsyn uses two INA169 current-shunt monitors [22] from TI to measure and amplify currents across 0.1Ω shunt resistors connected in series with high-side current wires/PCB-tracks that drive units of interest on the EMC2-DP. Measurements from the current-shunt monitors are sampled by a Teensy 3.6 microcontroller [21] using 13-bits and transmitted over USB to the host computer at approximately 12K samples per second. This rate supports measurements of

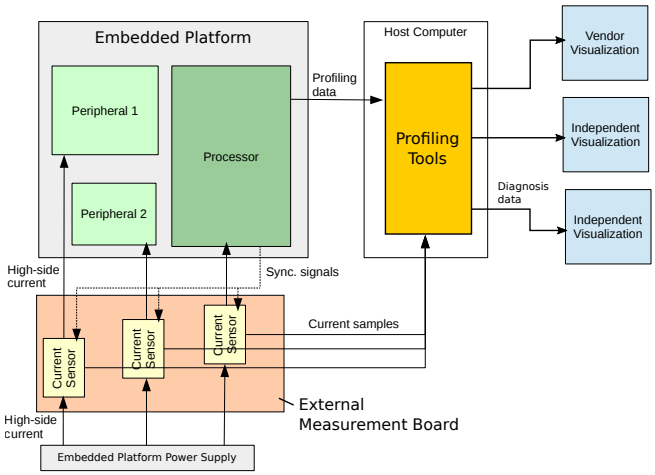

Fig. 3. Overview of power profiling

application tasks with runtime longer than 83 micro-seconds. Synchronization signals are sent over JTAG and LVTTL GPIO ports on the EMC2-DP to the Teensy to control measurements. At present, we consider two units of interest – the Zynq SoM and the FMC port that connects to the camera. Lynsyn can sense currents between 250 mA to 3 A. Measuring currents lower than 250 mA is possible by using larger shunt resistors. The BoM cost of Lynsyn is less than 50 US dollars. A protoboard version of Lynsyn connected to the EMC2 (non-stacked) is shown in Figure 4.

We validated the current measurements from Lynsyn using the Uni-T UT139C true-RMS digital multimeter as a reference for two hours of continuous operation. Current measurements had negligible differences compared to the reference. Rigorous testing with a constant current load is planned as part of future work.

Lynsyn's design assumes that it is possible to insert shunt resistors in all current-carrying lines of interest. However, not all current-carrying lines are accessible. For example, rails that supply power to the FPGA substrate of the Zynq SoM are buried due to dense packaging constraints. Potential workarounds include using current-mirrors to avoid inserting shunt resistors [13], or using special test fixtures that expose current-carrying lines on top layers [23].

Power visualizations: Current samples sent from Lynsyn to the host computer are converted to power readings assuming a constant supply voltage and stored in the Common Trace Format (CTF) [6] by a profiling tool. CTF is a flexible, high-throughput, binary trace format developed by the Multicore Association. The power traces can be visualized using Trace Compass, an open-source, standalone viewer popularized by the Linux Tracing Toolkit (LTTng) project [14]. Trace Compass enables correlation and filtering of power traces. An example is provided in Figure 5.

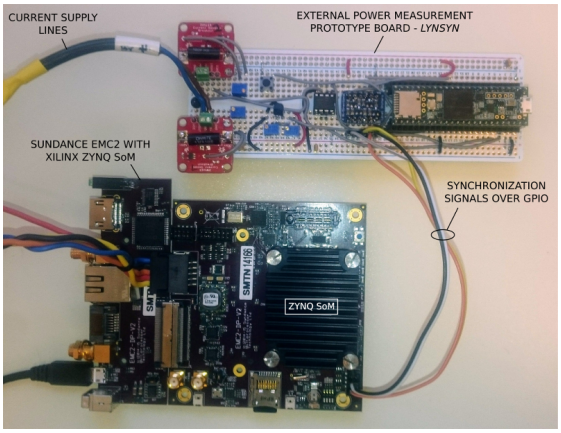

Fig. 4. Lynsyn, the power measurement board connected the EMC2-DP. The current supply wire connects to a current sensor. Synchronization signals are used to start and stop power profiling.

We visualize instantaneous power computed from the current samples on a running line graph as shown in Figure 6. This helps understand power trends in real-time as the application executes. Abrupt, large changes in power values are flagged on the visualization to alert users.

SDSoC enables users to understand timing of application events in a timeline visualization called the *AXI Trace Viewer* [25]. We extend the AXI Trace Viewer to visualize power traces correlated with application phases. This enables programmers to conveniently attribute power consumption to concurrent application events and isolate power problems. However, we are not able to refine user interaction in this mode since SDSoC is closed-source software.

Improving the PMU: As part of future work, we intend to profile application-specific data such as the program counter and parallelization events via the JTAG port while collecting power samples. The idea is to analyze this data to pinpoint power problems on high-level semantic visualizations such as control flow graphs and grain graphs [15].

2.2 HW/SW Image Processing Libraries (IPL)

The HW/SW Image Processing Library helps programmers implement accelerated image processing applications. A template-based software library for streaming based applications has been implemented (C++). FPGAs can outperform other hardware architectures, like CPUs and GPUs, for streaming based applications as shown in [11]. The provided functions have been optimized to be accelerated on FPGAs using SDSoC. Furthermore, the library has been optimized for latency, memory throughput and resource usage. The functions follow the OpenVX specification [12], to address a large group of users. OpenVX is an open, royalty-free standard for cross platform acceleration of computer vision

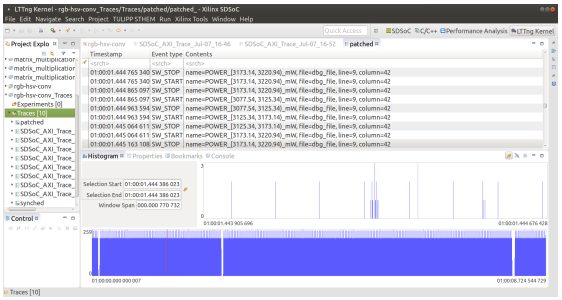

Fig. 5. Inspecting a power trace on Trace Compass

applications. Additionally, more data types and auto-vectorization are supported for most functions.

Normally, an image processing function processes one pixel per clock cycle. Using vectorization, it can process one, two, four or even eight pixel per clock cycle. The maximum bit-width of the complete vector is set to 64-bit. Therefore, the maximum vectorization depends on the bit-width of the image data. One advantage of vectorization is the possibility to process higher image resolution. Another advantage is that the frequency of the design can be reduced. Therefore, the power consumption of applications decreases.

The library contains several compile time optimizations, to reduce inputs from users. For example, the Gaussian kernel coefficients are computed at compile time using the standard deviation and kernel size. They are computed using double precision floating point numbers, then normalized and converted to fixed-point numbers. This computation does not consume extra resources of the FPGA logic. The developer will also get compile time errors if unsupported data types or combinations of them are used, to increase usability. Image data for functions can be in 8-bit, 16-bit or 32-bit fixed-point representation (unsigned/signed).

There are three groups of library functions. The first group consists of all windowed functions. This includes 3x3 Scharr, 3x3 Sobel, 3x3 Median, Box, Gaussian Convolution and Custom Convolution filters. All functions are normalized to avoid overflow (below 1.0 for unsigned and between 0.5 and -0.5 for signed) and optimized in their structure to reduce resource usage. The windowed operations support replicated, constant and undefined border handling.

The second group consists of all pixel-wise functions, which are basically bitwise and arithmetic operations. This includes: Absolute Difference, Arithmetic Addition, Arithmetic Subtraction, Gradient Magnitude, Pixel-wise Multiplication, Bitwise And, Bitwise Xor, Bitwise Or and Bitwise Not. The arithmetic operations support conversion policies against overflow and different rounding policies if needed.

The last group contains all remaining functions. This includes the Convert Bit Depth, Convert Color, Scale Down, Integral Image, Histogram and Table Lookup functions. The Color Conversion function can convert between the RGB, RGBX

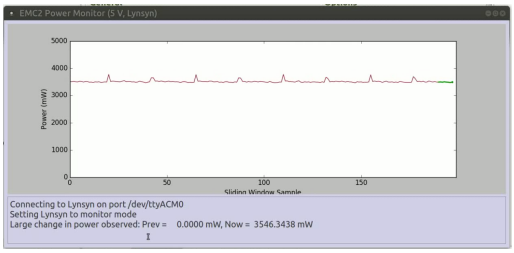

Fig. 6. Power monitor visualization tracks instantaneous power consumption during application execution

and grayscale formats. The Scale Down function supports nearest neighbor and bilinear interpolation.

2.3 Dynamic Partial Reconfiguration Utility (DPRU)

SoCs such as the Xilinx Zynq combine hardened processors with programmable logic which can be used to accelerate application hot-spots. The programmable logic can be partitioned into static and dynamic regions. The dynamic regions can be reconfigured at runtime while the logic in the static region is fixed. The procedure of reconfiguring the dynamic region is called *Dynamic Partial Reconfiguration (DPR)* [5, 3]. DPR allows upgrading the design without the need to erase the whole FPGA and saves programming time. Also, it allows more applications to be time-multiplexed onto the same FPGA.
The dynamic partial reconfiguration feature is used in TULIPP to:

- Reduce the need for FPGA-resources by fitting more functionality on the same set of programmable hardware.
- Reduce power consumption by disabling dynamic regions of the FPGA that are not used and re-operating them when they are needed.
- Runtime upgrading which enables more implementation techniques to be deployed at run-time.

DPR is being integrated into the STHEM utilities to allow the TULIPP platform user to update the design freely. Concretely, a set of TCL scripts have been developed to enable DPR within the Xilinx SDSoC high-level workflow [10]. These scripts extend SDSoC functionality for embedded application developments and add more options to the software-hardware partitioning task. In future work, we plan to add optimization logic that analyzes the design to help decide which parts of the application are implemented in static and dynamic regions.

2.4 HIPPEROS SDSoC Compatibility Layer (HSCL)

Xilinx SDSoC is a tool for developing applications for Xilinx System-on-Chip (SoC) architectures and enables the programmer to target its C/C++ code to one of the CPU cores or, through high level synthesis, to the FPGA fabric. HIPPEROS [7] is a multi-core real-time operating system (RTOS) that is adapted

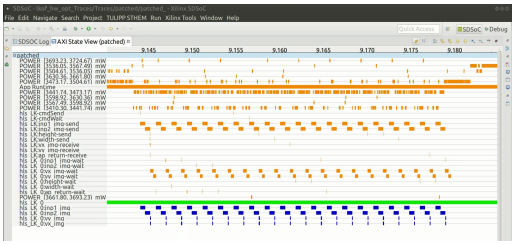

Fig. 7. Attributing power consumption to application events on SDSoC's AXI Trace Viewer

for high performance and safety-critical embedded systems applications [8]. The HIPPEROS SDSoC Compatibility Layer is added to Xilinx SDSoC to enable SDSoC to compile applications for HIPPEROS. Previous research work involving the HIPPEROS RTOS includes multi-core micro-kernel design [18], power-aware real-time scheduling [20] and mixed-criticality scheduling [19].

SDSoC is not easily extendable and supports only bare metal, FreeRTOS and Linux applications. Being a closed-source application, it is not possible for a third party to add an additional OS to SDSoC. The approach chosen in this project was to add HIPPEROS support under the disguise of being FreeRTOS.

To achieve this, the following components were necessary:

- *SDSoC platform description:* The SDSoC platform description is used by SDSoC to target a specific hardware platform. In addition to information about the available hardware, it also contains the necessary configuration files and libraries to compile for one of the three supported operating systems. HSCL adds to the platform description by modifying the FreeRTOS configuration files such that HIPPEROS binaries are used instead of FreeRTOS.
- *C library:* Both SDSoC and HIPPEROS need to be initialized correctly when the developed application boots. We cannot modify the SDSoC libraries, so the solution was to put all initialization code into a special C library that automatically gets linked to by the SDSoC toolchain. Additionally, SDSoC requires some specific ABI compilation flags to set for every object file linked within the accelerated program. Therefore, we created a specific HIPPEROS distribution dedicated to the Tulipp platform and the compatibility with SDSoC.
- *Scripts:* Unlike FreeRTOS, HIPPEROS needs an additional step after compilation to package the resulting elf file into an executable binary. The necessary script for doing this is provided and presented to the user in the SDSoC SD-card generation step.
- *Bootloader:* In order to correctly boot a HIPPEROS application, a bootloader is necessary. This is also the case when starting the application from the Xilinx debugger. It is not sufficient to upload the binary files to memory and jump to the entry address. Therefore, a small bootloader is provided

such that debugging and tracing from the SDSoC GUI or command line is possible.

3 Conclusion and Further Work

In this paper, we have presented the underlying philosophy of the analysis and development tools that will be developed during the TULIPP project. Further, we have described the implementation of our first four utilities: a novel power measurement and analysis utility (the PMU), a platform-optimized image processing library (the IPL), a dynamic partial reconfiguration utility (the DPRU), and an utility providing support for using the HIPPEROS RTOS within Xilinx SDSoC (the HSCL).

The work achieved so far forms the basis of the research that will be carried out during the second half of the TULIPP project. We will leverage the developed utilities to provide novel performance analysis and design space exploration tools that specifically focus on embedded image processing systems. In addition, we aim to quantitatively compare our image processing library to other libraries and full-custom FPGA implementations. Finally, we will use the utilities to improve the TULIPP use case applications. The use cases are industry-grade applications within the medical, automotive and unmanned aerial vehicle domains.

Acknowledgement The work is funded by European Commission under the H2020 Framework Program for Research and Innovation under grant agreement number 688403.

References

1. Abusultan, M., Khatri, S.P.: A comparison of FinFET based FPGA LUT designs. In: Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI. pp. 353–358. GLSVLSI '14, ACM, New York, NY, USA (2014)
2. Buschhoff, M., Günter, C., Spinczyk, O.: MIMOSA, a highly sensitive and accurate power measurement technique for low-power systems. In: Real-World Wireless Sensor Networks, pp. 139–151. Springer (2014)
3. Cornil, M., Paolillo, A., Goossens, J., Rodriguez, B.: Research and implementation challenges of rtos support for heterogeneous computing platforms. In: Heterogeneous Architectures and Real-Time Systems Seminar (May 2017)
4. Di Nisio, A., Di Noia, T., Carducci, C.G.C., Spadavecchia, M.: High dynamic range power consumption measurement in microcontroller-based applications. IEEE Transactions on Instrumentation and Measurement 65(9), 1968–1976 (2016)
5. Dye, D.: Partial reconfiguration of Xilinx FPGAs using ISE design suite. wp374 (v1.2) (2012)
6. EfficiOS: Common Trace Format (CTF). `http://www.efficios.com/ctf` (2017)
7. HIPPEROS: HIPPEROS Web Page. `http://hipperos.com/` (2017)
8. Jahre, M., Djupdal, A., Kalms, L., Muddukrishna, A.: D4.1: Basic Tool Chain. Tech. rep., TULIPP Project (2017)
9. Kalb, T., Kalms, L., Göhringer, D., Pons, C., Marty, F., Muddukrishna, A., Jahre, M., Kjeldsberg, P.G., Ruf, B., Schuchert, T., Tchouchenkov, I., Ehrenstrahle,

C., Christensen, F., Paolillo, A., Lemer, C., Bernard, G., Duhem, F., Millet, P.: TULIPP: Towards ubiquitous low-power image processing platforms. In: 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS). pp. 306–311 (Jul 2016)
10. Kalb, T., Göhringer, D.: Enabling dynamic and partial reconfiguration in Xilinx SDSoC. In: ReConFigurable Computing and FPGAs (ReConFig). IEEE (2016)
11. Kalms, L., Göhringer, D.: Exploration of OpenCL for FPGAs using SDAccel and comparison to GPUs and multicore CPUs. In: 2017 27th International Conference on Field Programmable Logic and Applications (FPL). pp. 1–4 (Sept 2017)
12. Khronos Vision Working Group: The OpenVX Specification (2017), https://www.khronos.org/registry/OpenVX/specs/1.2/OpenVX_Specification_1_2.pdf
13. Konstantakos, V., Chatzigeorgiou, A., Nikolaidis, S., Laopoulos, T.: Energy consumption estimation in embedded systems. IEEE Transactions on instrumentation and measurement 57(4), 797–804 (2008)
14. LLTng: Linux Tracing Toolkit Next Generation. http://www.lttng.org (2017)
15. Muddukrishna, A., Jonsson, P.A., Podobas, A., Brorsson, M.: Grain Graphs: OpenMP performance analysis made easy. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (2016)
16. Mukhanov, L., Petoumenos, P., Wang, Z., Parasyris, N., Nikolopoulos, D.S., De Supinski, B.R., Leather, H.: ALEA: A fine-grained energy profiling tool. ACM Transactions on Architecture and Code Optimization (TACO) 14(1), 1 (2017)
17. Nakutis, Z.: Embedded systems power consumption measurement methods overview. MATAVIMAI 2(44), 29–35 (2009)
18. Paolillo, A., Desenfans, O., Svoboda, V., Goossens, J., Rodriguez, B.: A new configurable and parallel embedded real-time micro-kernel for multi-core platforms. In: Proceedings of the ECRTS Workshop on Operating Systems Platforms for Embedded Real-Time applications (July 2015)
19. Paolillo, A., Rodriguez, P., Svoboda, V., Desenfans, O., Goossens, J., Rodriguez, B., Girbal, S., Faugère, M., Bonnot, P.: Porting a safety-critical industrial application on a mixed-criticality enabled real-time operating system. In: Proceedings of the 5th Workshop on Mixed-Criticality Systems (December 2017)
20. Paolillo, A., Rodriguez, P., Veshchikov, N., Goossens, J., Rodriguez, B.: Quantifying energy consumption for practical fork-join parallelism on an embedded real-time operating system. In: Proceedings of the 24th International Conference on Real-Time Networks and Systems. pp. 329–338. RTNS '16, ACM (2016)
21. PJRC: Teensy 3.6. https://www.pjrc.com/store/teensy36.html (2017)
22. Texas Instruments: INA169-Q1: Automotive Grade, 60-V, High-Side, High-Speed, Current Output Current Shunt Monitor. http://www.ti.com/product/INA169-Q1/datasheet/detailed_description\#SGLS1854308 (2017)
23. Trenz-Electronic: Test fixture for Zynq UltraScale+ MPSoC. https://wiki.trenz-electronic.de/display/PD/TEBT0808+TRM (2017)
24. Xilinx: Measuring ZC702 Power using TI Fusion Power Designer Tech Tip. http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Low+Power+Techniques+part+2+-+Measuring+ZC702+Power+using+TI+Fusion+Power+Designer+Tech+Tip (2014)
25. Xilinx: Xilinx Environment Tutorial (UG1028) (2016)
26. Xilinx: Xilinx Power Estimator. https://www.xilinx.com/products/technology/power/xpe.html (2017)
27. Xilinx: Xilinx XADC User Guide. https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf (2017)