Parallel Ultra Low Power Embedded System

João Pedro Alves Vieira
joaopvieira@tecnico.ulisboa.pt
Instituto Superior Técnico, Lisboa, Portugal

Abstract— **The future of portable electronics' market will be built around Internet of Things (IoT), where everyday objects will be connected to the internet and possibly controlled by other devices. In fact, examples of these devices have already started to take part on our daily activities and are expected to experience a tremendous growth in a near future, such as health monitors, light bulbs, thermostats, fitness wristbands, etc. Most of these devices rely on battery-powered wireless transceivers combined with sensors, where it is essential to sustain energy-efficient execution by developing devices' architectures capable of delivering both low power and real-time computing performance. Within the scope of IoT applications, this Thesis aims to boost the energy-efficiency of a state-of-the-art ultra-low-power processor, namely PULPino. This challenge was tackled by modularly attaching hardware accelerators to it. They connect to PULPino through a low-power and plug-n-play custom AXI-lite interface. It has the objective of encouraging the development of new accelerators by the growing PULPino's open-source community. To test the viability of this approach, two kinds of accelerators were individually attached. A first cryptographic SHA-3 accelerator, implementing a commonly used hash algorithm, that could improve IoT applications' security. And second, an FFT accelerator, having a widely used algorithm in Digital Signal Processing (DSP) applications. Both accelerators were tested on PULPino, for their speedup and energy-efficiency capabilities. Achieving savings up to 99% and 66% of energy, speedups of 185 and 3 times on SHA-3 and FFT respectively. In comparison to a non-hardware accelerated version of the algorithms executed on PULPino RI5CY core configuration.**

I. Introduction

In order to satisfy the growing demands of current consumer electronics market, it is estimated there will be around 50 Billion of Internet connected devices (”things”) by 2020. A big part of IoT lies on battery-powered wireless transceivers combined with sensors (also known as motes). Such devices demand ultra-low-power circuits and are usually controlled by a Microcontroller (MCU) responsible for sensor interaction and light-weight processing. In IoT topology, besides the basic motes basically gather data, there are also the end nodes that can be configured as gateways for the motes, which allows gathering data from the motes and eventually pre-processing it before sending to the cloud. One solution for end nodes is presented in this Thesis, i.e. PULP in Section II-A, which is able to satisfy both computing and energy-efficiency requirements of IoT applications, by taking advantage of parallel computing. Based on PULP solution, its further simplification into a more basic unit (PULPino) that fits the motes' requirements will be the main goal of this Thesis, in order to provide an adequate substitute for the MCU.

IoT nodes might be optimized in many different ways achieve enhanced energy-efficiency and performance. One solution goes by attaching hardware accelerators to the IoT node. Under the scope of this Thesis, were chosen well known types of applications in which IoT node might benefits from. Such as, Digital Signal Processing (DSP) and cryptography. DSP applications include a wide variety of kernels which are recursively executed, and might require considerable computational power. Cryptography on IoT systems is a recently hot topic. Due to the reduced computational capabilities and low-power characteristics of such systems, having the required security might be very challenging. To achieve it, not only is needed a reduction of computational overhead of cryptography algorithms. But also, a reduction of its associated extra energy consumption.

To tackle such challenges and move a step further to these objectives, were developed hardware accelerators, one for each application (DSP and cryptography), based on a modular approach and a common interface with the processor. Encouraging the open-source community to develop and share their own custom IPs. Continuing a path to open-source hardware, as the release of the further presented PULPino was intended for.

Chapter 2 (Background) provides an overview of PULP and PULPino features/related work. Also addressed emerging interconnect networks and relevant applications of hardware accelerators.

Chapter 3 (Hardware/Software Co-Design) explains the hardware/software architecture, developed with the purpose of attaching the hardware accelerators into PULPino's main interconnect bus.

Chapter 4 (Implementation and Experimental Work) describes how to set up an working environment, as how to use the developed AXI-lite interface.

Chapter 5 (Analysis and Experimental Results) contains an analysis of the obtained experimental results.

II. Background

In this chapter will be presented the state-of-the-art and related work done with PULP, a novel cluster platform intended to be released as open source hardware in 2018. Since PULPino represents a small part of PULP (one core), which has been already released and is a main focus of this Thesis. Since the attached hardware accelerators interface through an interconnect bus, state-of-the-art interconnect networks are addressed. Also an overview on the state-of-the-art implementations of hardware accelerators is presented.

A. State-of-the-Art: PULP - Parallel Ultra Low Power Platform

PULP development team aims to develop an open source scalable hardware and software platform with the objec-

tive to break the pJ/op barrier within power envelopes of a few mW [1]. PULP's architecture is tuned for efficient near-threshold operation, overcoming the power constraints of battery-powered applications, which are restricted to a power envelop of a few mW [2]. A PULP cluster embeds a configurable number of RISC-V based cores with a shared instruction cache and scratchpad memory. PULP achieves an energy efficiency of 193MOps/mW [3]–[5]. The performance of its 28nm FDSOI implementation can be adjusted up to 2GOPS by scaling the voltage from 0.32V to 1.15V with the cores operating at 500MHz [3]–[5].
The PULP cluster is perfectly suited for IoT endpoint devices due to its efficiency and low power consumption while still keeping high computational power [6]. Only one core from PULP, named PULPino is available at the moment under open hardware license, it can be intended for IoT remote nodes that do not require as much computational power as an endpoint device. PULP has been subject of several scientific works, briefly presented next, targeting different application areas and architecture extensions, although solely on PULPino there are no scientific work, up to now.

Computer Vision Applications

PULP has been used in a set of applications regarding energy-efficient computer vision by taking advantage of its parallel computing and support for OpenMP [7]. In [8] it is shown that a computationally demanding vision kernel based on Convolutional Neural Networks (CNN) can be quickly and efficiently switched from a low power, low frame-rate operating point to a high frame-rate one when a detection is performed. Scaling PULP's performance from 1x to 354x, reaching 211 GOPS/W.

Extensions and HW Acceleration

Hardware extensions were also explored in order to bring new energy-efficient solutions, e.g., in the case of a shared Logarithmic Number Systems (LNU) unit implemented in [9], which was as an energy-efficient alternative to a conventional Floating Point Unit (FPU) . This LNU, optimized for ultra-low-power operations on PULP multi-core system, is efficiently shared by all the cores. For typical nonlinear processing tasks, this design can be up to 4.2x more energy-efficient than a private-FPU design.

Heterogeneous Programmable Accelerator

PULP was used also as a heterogeneous accelerator for speeding-up computation-intensive algorithms. In [10], a heterogeneous architecture was developed by coupling a Cortex-M series MCU with PULP, supports offloading of parallel computational kernels from the MCU to PULP by taking advantage of the OpenMP programming model, supported by PULP.

B. PULPino

PULPino is a small single core system based on PULP, as previously mentioned in Section II-A. As such, PULPino being the main focus of this Thesis, represents a first step towards the release full of PULP as an open-source platform.Its open-source release was in February 2016 under the solderpad hardware license, including complete RTL sources, all IPs, RI5CY core based on RISC-V, environment for RTL simulation and the complete FPGA build flow. PULPino features an IPC close to one, full support for the base integer instruction set (RV32I), compressed instructions (RV32C) and partial support for the multiplication instruction set extension (RV32M). Non-standard extensions have been implemented featuring hardware loops, post-incrementing load and store instructions, ALU and MAC operations. Dot-product and sum-of-dot-products instructions on 8b and 16b data types allow to perform up to 4 multiplications and accumulations in one cycle, consuming the same power as a 32b MAC operation. A low power mode is available, being able to wake up in case of an event or interrupt arrival. Once in low power mode, only the event unit is active and everything else is clock gated, consuming minimal power due to leakage.

Pipeline Architecture

The organization of the pipeline is illustrated in Figure 1, which consist in a four stages: instruction fetch (IF), instruction decode (ID), execute (EX) and write-back(WB). It was possible to extend the ALU with fixed-point arithmetic and enhanced multipliers that support *dotp* operations (while still keeping the same timing), since the critical path is mainly determined by the memory interface. The cluster can achieve frequencies of 350-400MHz, being able to reach higher frequencies than the commercially available MCUs that usually operate in the range of 200MHz [6].

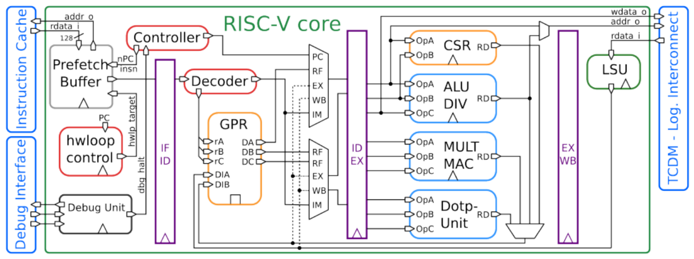

Fig. 1. Simplified block diagram of the RISC-V four stage pipeline [6].

Additional PULPino's Core Configurations

Despite the initial release of PULPino dating February 2016, which is based on the previously presented RI5CY architecture, additional core configurations were release in August 2017. Featuring three newly available cores:

- **RI5CY + FPU:** the same previously presented RI5CY core enhanced with a single precision Floating Point Unit

(FPU), compliant with IEEE-754 standard for floating-point arithmetic.

- **Zero-riscy:** An area-efficient 2-level pipelined core, implementing RISC-V RV32-ICM instruction set as RI5CY core.
- **Micro-riscy:** An even smaller core than the previous ones, implementing RV32-EC instruction set. Having only 16 general purpose registers and no hardware multiplication support.

C. Interconnect Networks

Computer based systems have a need to connect its individual components together and allow them to communicate with each other like a community. These networks rely on communication standards to establish rules of how the data will be converted and transfered among several components. Network-on-chip (NoCs) are used for interconnecting micro-architecture functional units within chips. Recent standards, purpose an improvement on interconnectivity between accelerators as other system components(e.g. CCIX, Gen-Z and OpenCAPI) [11]. These three new standards, were announced in 2016 being developed towards the goal of optimizing and easing the connection between accelerators and processors in a tightly-coupled manner. These will be explained in more detail up front.

Cache Coherent Interconnect for Accelerators (CCIX)

CCIX was founded with the purpose of enabling a new class of interconnect, based on emerging acceleration applications. It allows processors based on different ISAs to peer processing to multiple acceleration devices. It uses a tightly coupled interface between processor, accelerators and memory, together with hardware cache coherence across the links [12].

GEN-Z

GEN-Z defines itself as a high-performance, low latency, memory-semantic fabric enabling communication throughout every device in the system. Creating an ecosystem in which a wide variety of high performance solutions can communicate together, by unifying communication paths and simplifying software through load and store memory-semantics. [13]

Open Coherent Accelerator Processor Interface (OpenCAPI)

OpenCAPI is an Open Interface Protocol that allows any processor to attach to coherent user-level accelerators, I/O devices and advanced memories (accessible via read/write or DMA semantics). The semantics used to communicate with the multiple components are agnostics to processor architecture. The main key attributes are high-bandwidth, low latency, based on virtual addresses implemented on the host processor to simplify the attached devices [12].

III. Hardware Acceleration

An hardware accelerator is a specialized unit designed to perform a very specific task or set of tasks, achieving higher performance and energy efficiency than a general purpose CPU unit in such specific application. While a co-processor executes instructions dispatched by the CPU, an accelerator is a device attached to the bus, controlled by registers. [14]. Accelerators are good for real-time applications, I/O processing, data streaming, specific ”complex” functions (DCT, FFT, etc) or specific ”complex” algorithms such as neural networks. Designing such systems in hand-written RTL implementations is highly tedious, time-consuming and consequently costly.A possible and attainable solution is presented in this Thesis, by defining an light-weight interface based on axi-lite standard, between the processor and accelerator. Allowing the reuse of accelerators hardware designs in any system that supports the AXI specification. Allowing the reuse accelerators in different scenarios. The speedup an accelerator might provide is based on the equation of accelerator’s total execution time:

$$T_{acc} = t_{in} + t_{comp} + t_{out},$$

In which t_{in} and t_{out} represent the time it takes to transfer the input and output data respectively into and out of the accelerator and t_{comp} for the accelerator’s computing time. Cryptographic and Digital Signal Processing (DSP) functions are highly suitable for hardware acceleration, due to its frequent use, requiring a considerable number of operations per input, are generally inefficiently when computed in general purpose CPUs. Thus, improving its performance and overall energy consumption. For instance, an SHA-3 specialized hardware accelerators was proposed in [15]. The accelerator implemented as a co-processor compliant with ROOC interface, is based on an parametrized implementation using automated tools and integrated with a Rocket RISC-V processor. Besides security, Digital Signal Processing (DSP) is as well a huge subject on which most of nowadays computing intensive applications are based on.Some of these applications are currently used in novel IoT or ultra-low-power systems, as targeted in [8], [16], [17], [7] and [18] regarding computer vision as presented in Section II-A. Today’s main FPGA manufacturers already provide full featured DSP cores, for FFT take for instance [19] [20] from Xilinx and Intel, respectively. More optimized solutions and claiming to beat such proprietary core implementations arrived. SPIRAL a novel hardware generation framework and system for linear transforms is introduced in [21]. It will automatically generate an algorithm, mapping it to a datapath and finally results an synthesizable RTL verilog description file, which is ready for FPGA or ASIC implementation.

IV. Hardware/Software Co-design

A. AXI Protocol

The AXI protocol enables the main system components on PULPino to be memory mapped. Creating the possibility to access those components through the core with simple load/store instructions.

AMBA AXI4 is an open standard specified by ARM. Facilitating connection and management of functional blocks in SoC designs. It is now *de facto* standard for embedded processors due to its free royalties and well documented specifications. It encourages modular systems to be reused across different systems and applications. While maintaining an high performance and low power communication. AXI4 has several subsets, namely AXI-full, AXI-lite and AXI-Stream.AXI-full targets high performance, high clock frequency systems designs.The protocol operates in a master-slave paradigm, meaning that each end of the connection obligately has to be either a master or a slave. It uses 5 different channels: read address; read data; write address; write data and write response.AXI-lite is a lighter implementation of AXI-full protocol. Uses the same 5 channels. Each transfer is limited to a data width of 32 or 64bits. Is suited for simple implementations that do not have severe bandwidth requirements.Supports from 4 to 512 individual addressable slave registers. Each of them may be written to or read from. With its simpler implementation comes a smaller footprint, advantageous in ultra low-power systems in which every saving matter. Despite its reduced performance, its possible to bridge back to AXI-full. Allowing a interaction between both protocol specifications, even creating with ease a bridge among low and high throughput systems. AXI-Stream makes use of one data channel in which the data only flows in one way: from master to slave. Designed to applications that require high bandwidth data transfers and low latency. Having only a data channel and unlimited burst of data, does not require addresses to proceed with the transactions. AXI-full is used in the current PULPino's design to connect together components. Other components, like instruction and data memories, core and peripherals make use of AXI-full indirectly, through an AXI interconnect. AXI-lite, being a much simpler and lightweight protocol, in this Thesis design it is intended to handle communications between core and accelerator, not only for data transaction, but also for control purposes.

B. AXI Interconnect

On an overview, Pulpino integrates multiple components, resorting to an interconnect network. As depicted in Fig.2, it uses a main interconnect AXI block and a bridge to Advanced Peripheral Bus (APB) to connect simple peripherals. Both featuring a 32bit wide data channels. Pulpino has its components connected through an AXI interconnect block allowing, all them to be mapped in a memory space. Providing a homogeneous view of the system. A memory map has been defined, in which all the components have user-configurable address spaces.If an accelerator needs to be added, a new set of addresses (start and end) is configured at the AXI interconnect configuration sources. The interconnect block provides a way of multiple masters and slaves to be connected to several blocks at once.

C. Overall System Architecture

This section addresses the proposed overall system architecture, based on PULPino. In order to improve PULPino's energy efficiency, hardware acceleration was provided in an loosely-coupled manner. The goal is to develop a generic plug and play interface for the accelerators to be easily attached to the core (Section IV-D), without having to adapt its interface each time a different one needs o be added. The processor is able to interact with the accelerator by simple load/store instructions. Through which the processor-accelerator synchronization mechanisms are implemented, further detailed on Section IV-E. The accelerators are connected interfacing with the processor through an AXI connection. The accelerator interfaces with an AXI-lite to AXI-full converter, as shown in Fig. 3. The AXI-full interface of the converter is connected to PULPino's AXI interconnect block, that handles the communications coming from the processor.

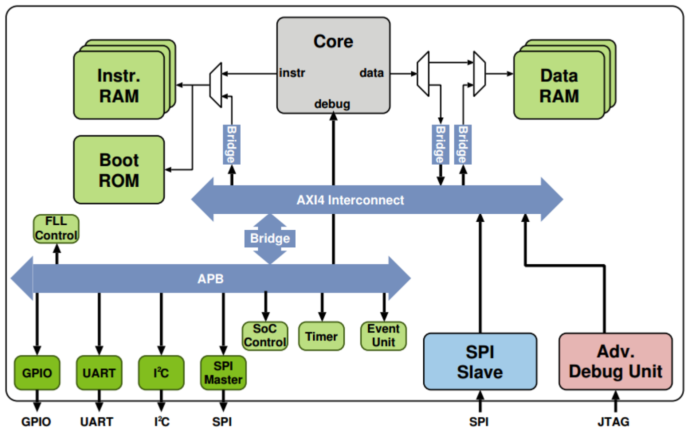

Fig. 2. PULPino's SoC block diagram [22].

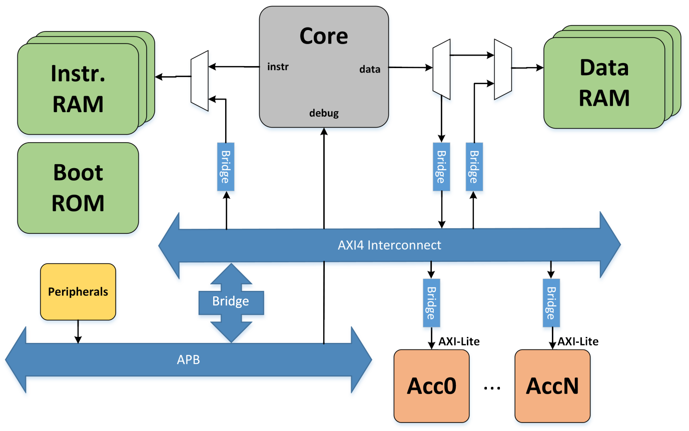

Fig. 3. PULPino with attached accelerators block diagram.

D. Hardware interface

The hardware interface between accelerator and the remaining system is intended to be loosely-coupled. The accelerator block only requires an AXI-lite port as interface, see Fig. 3. This block behaves like a wrapper that integrates the kernel and has all the required hardware to handle the interface between it and the AXI-lite slave registers. The AXI-Lite interface that connects to the accelerator has the following configuration: 2 bits address width; 32 bits data width; Read-/Write modes enabled. After the accelerator is connected to the AXI interconnect block, it is memory mapped and addressable by the processor. Each accelerator to be plugged in, has to have an independent memory region configured in the AXI interconnect block and its own protocol converter, as depicted in Fig. 3. Both the converter and accelerator are instantiated in *core_region.sv* top module. To connect the accelerator to the converter, a new AXI4 slave bus was instantiated.The AXI slave port of the converter is connected to the AXI interconnect block's master AXI port. On the other hand, accelerator's AXI-lite slave port connects to the AXI-lite master port of the converter.

E. Software Interface

The processor interacts with the accelerator through load/-store instructions, addressing the memory region predefined to it. In essence the processor needs to send the data to be processed and afterwards fetch the computed results. To synchronize accelerator-processor was defined communication protocol, based on read/writes using the AXI-lite slave registers, described on Table I. To reset the accelerator, a write to

TABLE I
AXI-LITE SLAVE REGISTER WRITE/READ MAP FUNCTIONALITIES.

Register	**Write**	**Read**
slv_reg0	Reset	Done
slv_reg1	Input Data	N.A.
slv_reg2	Last Data	N.A.
slv_reg3	Optional	Result

its first address with the value 0x01010101.Afterwards, data can be streamed into address 1. The last data value should be sent to address 2Once the computation is done, the value *0xdeadbeef* can be read from address 0, meaning the output values are available to be fetched in address 3. An optional functionality for *slv_reg3* was added to fill some extra need that some accelerators might have.

V. HARDWARE ACCELERATORS

The hardware accelerators were chosen upon the first criteria of being open source.The kernels used were not inner modified, upon its requirements was developed the required hardware to accommodate them within the accelerator block. On the following subsections are presented two accelerators. Due to different kernel's requirements, different hardware designs were needed for both.

SHA-3

An SHA-3 accelerator aims to provide a faster computation of the hash function with a reduced energy cost, for the addressed low-power processor. The input message might take any size, that the output length n will remain the sameThe current implementation has the highest security level of 512 bit among all SHA-3 variants.

The kernel was developed by Homer Hsing, available in OpenCores website is under the Apache license (version 2) [23]. Capable of computing an 512 bit hash result in 29 clock cycles, is based on a padding module followed by a permutation module. The kernel presents input/output ports shown in Table II.
To start computing a hash value, the core must be reseted by holding the reset signal synchronously high during one clock cycle. This procedure must be repeated at every new hash value computation,

TABLE II
TABLE OF SHA-3 KERNEL'S INPUT/OUTPUT PORTS.

Port	**Width**	**Direction**	**Description**
clk	1	In	Clock
reset	1	In	Synchronous positive asserted reset
in	32	In	Input data
byte_num	2	In	Number of bytes of *in*
in_ready	1	In	Input is valid or not
is_last	1	In	Current input is last or not
buffer_full	1	Out	Buffer is full or not
out	512	Out	Hash result
out_ready	1	Out	Result is ready or not

To comply with the kernel requirements, was developed the following data path is depicted in Fig. 4. The design aims

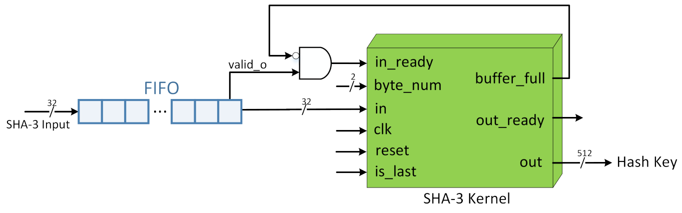

Fig. 4. SHA-3 accelerator data path.

for efficiency and reduced hardware usage, using only one FIFO (32 words of 32bit) and the required control signals. The input data from the processor is feed into the kernel as soon as it arrives (every 11 clock cycles). The FIFO acts as a buffer in the event of the kernel's buffer achieves full capacity and asserts the *buffer_full* signal. When the last input data is sent by the processor, FIFO might have exceeding data due to previous *buffer_full* events. Such data is streamed into the kernel on every clock cycle after the last input data. Which is sent to the *slv_reg2* AXI-lite register, asserting the *is_last* and selecting the previously received *byte_num* value. As before explained, the byte_num signal has to follow certain rules, based on the length of the last message block. The user is responsible for this handling, by sending a last dummy input data with the value zero after the last value message block, into the *slv_reg2* register. When the out_ready signal is asserted, the 512bit hash key is made available at the output port, being iteratively fetched by the processor on blocks of 32bit. This output is redirected to the *slv_reg3* register, since it is limited to a 32bit word, an auxiliary counter is used to iterate through the 512bit result.

FFT

Fast Fourier Transform (FFT) is widely used in DSP and in many other field of applications as a fast and more efficient algorithm to compute the Discrete Fourier Transform (DFT). The kernel chosen for this accelerator is based on an open source licence, as the previous one. Only its integration within a wrapper, was developed from scratch. Interfacing with the processor through an AXI-lite interface. All the additional hardware was dimensioned with the goal of reducing the used hardware. Its main goal is to adapt between both data input bandwidths. In this case, the AXI-Lite interface is the one to introduce limitations.

The chosen FFT kernel was developed by SPIRAL - Software/Hardware Generation for DSP Algorithms [21]. SPIRAL provides an online tool [24] for hardware generation. Table III, specifies the parameters chosen for the used kernel. It was chosen a kernel with 256 number of samples n, for the forward DFT defined as:

TABLE III
SPIRAL FFT KERNEL ONLINE CONFIGURATION PARAMETERS [24].

Parameter	Value	Range	Description
Problem specification			
transform size	256	4-32768	number of samples
direction	forward		forward/inverse DFT
data type	fixed point		fixed/floating point
	32	4-32 bits	fixed point precision
	unscaled		scaled/unsaled mode
Parameters controlling implementation			
architecture	iterative		iterative/fully streaming
radix	2	2, 4, 16	size of DFT block
streaming width	2	2–256	complex words per cycle
data ordering	natural in/out		natural/digit-reversed
BRAM budget	no limit		maximum BRAMs
Permutation method	[25]		[25] or [26]

$$y = DFT_n x,$$
$$DFT_n = [e^{-2\pi jkl/n}]_{k,l=0,\ldots,n-1}$$

where y is the n point output vector, and x the n point input vector. The data type chosen was ”fixed point”, due to the lack of floating point operations support by the processor. The current AXI-Lite configuration is set to 32 bit messages, the fixed point precision was set to 32 bit together in unscaled arithmetic mode. The architecture can be both ”iterative” or ”fully streaming”, since the developed wrapper for the Thesis supports both. ”Iterative” consists on a single stage architecture, requiring less hardware. Consequently is slower than the ”fully streaming” version in which the data stream would flow in and out the system continuously.
The FFT kernel given the previous configurations was wrapped, using the following data path depicted in Fig. 5 .

It is ready to handle different FFT kernel's configuration with minor adjustments on the input accelerator's input parameters, which are the following:

- FFT_INOUT_NR: kernel's number of inputs/outputs.
- DATA_WIDTH_FIFO: The data width of FIFO.
- DATA_DEPTH_FIFO: Data depth of FIFO.

VI. Implementation and Experimental Work

As stated before, PULPino is an open-source project, therefore all the basic material was retrieved from the project's

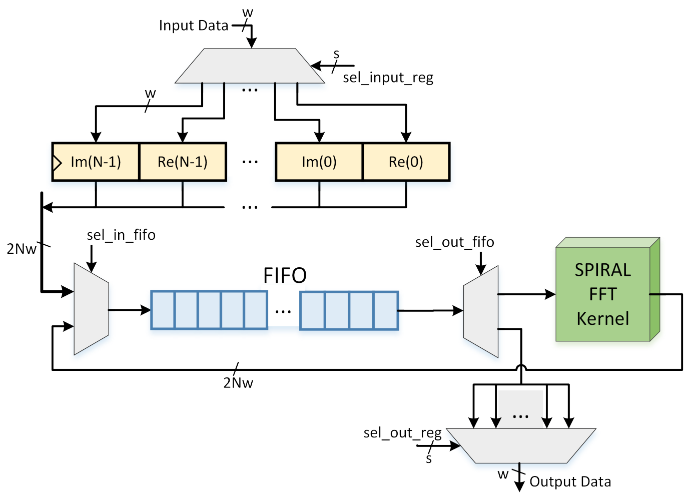

Fig. 5. FFT accelerator's data path.

GitHub page [27]. Knowing that this was its first release, and not a mature project, its expected to have incompatibilities and unsolved issues/bugs. This was one of the main setbacks found. Having to deal with a release that was not well documented from a technical point of view (only a basic user manual and a datasheet are available). Due to the recent release (dating 2016), there is still a very small active open source community working with this platform. Difficulting the resolution of the many of prompt issues.

A. Target Device

Pulpino is mainly targeted for RTL simulation and ASICs, although there is also a FPGA version supported on ZedBoard. being the development board chosen. The FPGA version is not optimal in terms of performance and efficiency, since it was used mainly for emulation instead of standalone platform. ZedBoard carries a Xilinx Zynq-7000 Family All Programmable System on Chip (SoC) XC7020-CLG484-1.

The main components are: Programmable Logic (PL) and Processing System (PS). The PL is derived from Xilinx 7 series FPGA technology: Artix-7 for the present XC7020 device. The integrated PL block is available for the user to program its own custom designed hardware.The PS itself features an Application Processing Unit composed by two ARM cortex A9 *hard-cores*.Available to the PS through central interconnect block, are I/O peripherals inner composed of SPI, CAN, UARTs, I2C, USB and Ethernet interface.

B. Adding a New Accelerator

The accelerator is instantiated in the core_region.sv file, wherein all the core related hardware blocks are as well (debug, data and instruction memories, protocol converters, memory multiplexers and RISC-V core). In the same core region, the accelerator, holding an AXI-lite slave interface is connected to the master interface of the AXI-full to AXI-lite converter. To do so, on the same file, should be added in the AXI interconnect instantiation, the address region for the AXI-lite converter. The memory map should be consulted in order to allocate it to an available address region. The new accelerator inputs all the AXI-lite signals, having all the necessary logic handle them.

VII. Experimental Results

A. Software vs Hardware

To measure the speedup that an hardware accelerator provides against software-only one, where set test benches to verify the performance and energy efficiency of both implementations. Herein this section are presented the software-only algorithms (which do not require an accelerator), and the ones that interact with the accelerators, used to compute SHA-3 and FFT. The software-only algorithms were adjusted to perform in the most similar way to the accelerators implementation of the kernel. Both accelerators were synthesized and implemented with the same tools and optimization strategy.

B. SHA-3

The algorithm used to implement SHA-3 was based on [28] implementation. It was configured to resemble the accelerator functioning. The base input test message used was: ”The quick brown fox jumps over the lazy dog ”. Having a size of 44 bytes, is a commonly used test message on different kinds of hash and encryption algorithms.

A set of tests were performed with different sized messages, always with the same message as before, but replicated up to 10 times. After running the tests to evaluate the amount of clock cycles required for both implementations, with a single 40MHz clock (maximum frequency on FPGA), is possible to verify the speedup in Figure 6.

Being possible to verify a speedup up to 185 times on a 440

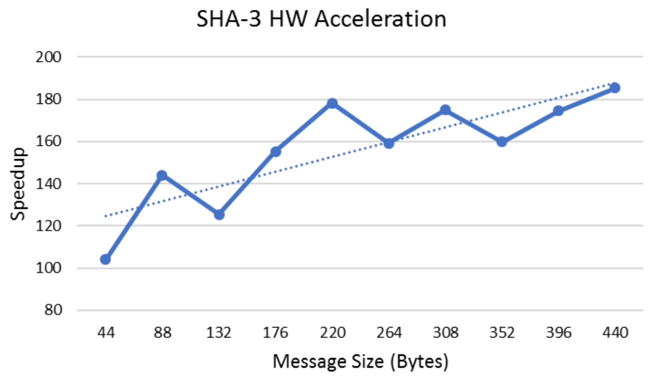

Fig. 6. SHA-3 computation speedup using hardware accelerator. Multiple message sizes were tested.

bytes length message, in comparison to the non accelerated version. With the speedup tending to increase along side with the input message length.

C. FFT

A well known algorithm (Cooley-Turkey) was used to test the performance of the FFT on software-only, which implementation was based on [29]. Unfortunately, the hardware developed presented in Section V, implementing the FFT accelerator, is only fully functional in simulation. Although, the hardware was properly mapped after synthesis and implementation. Making it possible to estimate the power consumption. Both speedup and power results are still valid and comparable with the SHA-3 accelerator, relying on the same tools, optimization settings and platform. A data set of 256 complex samples were defined as input. Having a total of 2K bytes of input and output data.
Multiple setups were tested by changing the kernel’s configurations. To test the FFT accelerator, was developed an algorithm which allows the core to interface with it.

With the algorithms and by changing the kernel’s parameters (architecture and radix, Table III). The architecture may vary between iterative and streaming version. The radix configurations were the ones presented in Figure 7. In which is shown the speedup achieved. The software-only FFT algorithm performs on a total of 38126 clock cycles, on which were based the speedup calculations. The speedup can be interpreted as the ratio between the amount of clock cycles required to conclude computation, with and without hardware acceleration. The stream version speedup overcomes the iterative one, although it is achieved with extra hardware cost. Not so significant

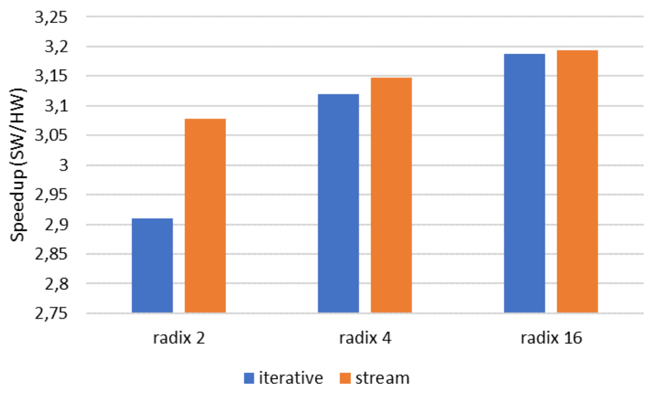

Fig. 7. FFT computation speedup using hardware accelerator. On multiple radix implemented in iterative or stream mode.

speedup results were achieved with FFT, when compared to SHA-3 previous analysis. This is in part due to the increased amount of compressed instructions used in FFT algorithm, in comparison with the SHA-3 one. The FFT software only algorithm translates into 87% compressed instructions. On the other hand the SHA-3 software-only algorithm has only 26% of compressed instructions. RISC-V compressed instruction claim to increase, not only performance, but also energy-efficiency and reduce the code size [30].

D. Power Efficiency

For an energy efficient hardware acceleration, the power savings achieved by entering faster in sleep mode, need to overcome the extra static energy cost accelerators bring over. Due to the required development platform (ZedBoard) restrictions, is not possible to measure real power consumption on the FPGA fabric alone. Only to estimate it by using the available tools, explained further ahead. To overcome this issue, would be needed a development board, that could power the FPGA chip with an external power supply. Additionally, would also be needed real time control of the FPGA's package temperature, due to its influence in power consumption measurements [31]. Having such hardware restrictions, the setup used to measure the system's power consumption was Xilinx Vivado working together with Modelsim as simulation tool.

E. SHA-3

SHA-3 accelerator energy efficiency was tested by performing multiple encryptions, with different message sizes. They were performed in a PULPino implementation with and without hardware accelerator. Not having any additional hardware when testing the software-only, in order to obtain most accurate power measurements. On Figure 8, are depicted the power measurement results obtained. In which are shown the computation power consumption, corresponding to the total amount of energy required to perform the message encryption. Notice that the total amount of energy required by Pulpino, with SHA-3 accelerator, is multiplied by a factor of 10, in order to be perceptible in the graph. Both were calculated accordingly with the following equation 1:

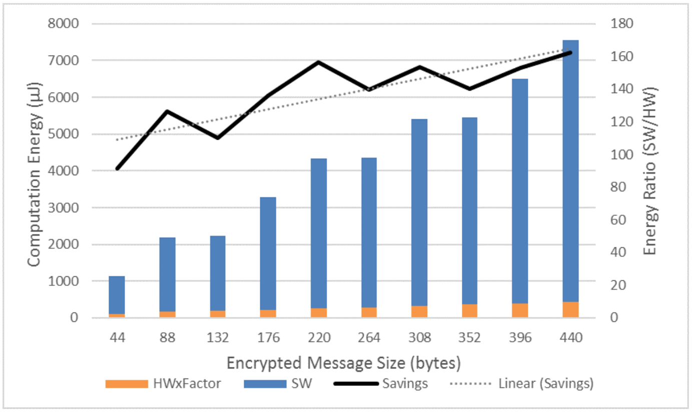

Fig. 8. Pulpino with SHA-3 accelerator computation energy with and without hardware acceleration. Combined with achieved energy ratio (SW/HW) at 5MHz

$$ComputationEnergy = \frac{1}{Freq} * ClkCycles * Power \quad (1)$$

In which $Freq$ corresponds to Pulpino's main operation frequency, $ClkCycles$ to the amount of clock cycles required by the processor to conclude the computation and $Power$ defines the on-chip power consumption estimated by Vivado. This power result is composed by two main parcels, dynamic and static power. The dynamic power is originated from the logic switching activity. The static power translates the power consumed by the FPGA logic when no signals are toggling. Regarding PULPino with SHA-3 accelerator, the dynamic and static values of on-chip power consumption are 189mW and 123mW, respectively. On the stock version of PULPino, the dynamic and static values are lower, 47mW and 121mW, respectively. This is due to not having the additional hardware that the accelerator bring on. Although, is possible to notice, that the additional static power is very small. Adding only 2mW, which translates into a 1.7% increase on static power On the right vertical axis of Figure 8, is represented the energy ratio, depicted as the black full line on the graph. Which corresponds to the amount of times the energy required to compute was reduced, by using the accelerator. At 5MHz, it is reduced up to 160 times when the message length is 440 bytes. As the tendency line of the energy ratio indicates, higher energy-efficiency can be achieved, as the message length increases. The same pattern is common among all remaining frequencies results. Although, the maximum achieved energy ratio varies. At 20MHz a maximum energy ratio of 114 times, was achieved. At 40MHz, it can only go up to 100 times. All of this values correspond to an input message of 440 bytes length.

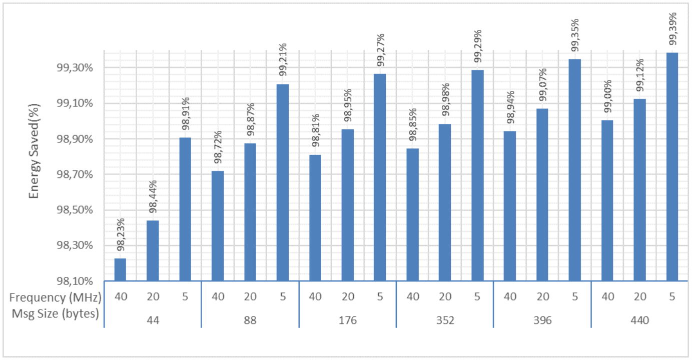

Fig. 9. Pulpino with SHA-3 accelerator energy saved. At 40MHz, 20MHz and 5MHz of main clock frequency with different encrypted message sizes.

Msg Size (bytes)	Frequency (MHz)	Energy Saved (%)
44	40	98,23%
44	20	98,44%
44	5	98,91%
88	40	98,72%
88	20	98,87%
88	5	99,21%
176	40	98,81%
176	20	98,95%
176	5	99,27%
352	40	98,85%
352	20	98,98%
352	5	99,29%
396	40	98,94%
396	20	99,07%
396	5	99,35%
440	40	99,00%
440	20	99,12%
440	5	99,39%

On Figure 9 is shown how much energy, in percentage, is saved in comparison with the stock version of PULPino, without hardware acceleration. Energy savings go up to 99.39% on at 5MHz with a message size of 440 bytes. On lower frequencies the energy savings are higher, has stated. The operation frequency, tends to have less impact on the energy saved, for longer message sizes. Meaning, that "long" messages can be computed faster, by increasing the main clock frequency, with less impact on energy savings.

F. FFT

On the pulpino with FFT accelerator, the same input was tested on multiple configurations of the FFT accelerator's architecture. On Figure 10 is shown the total on-chip power consumption in mW, divided in two parts: dynamic and static power. Each column of the graph represents a new hardware configuration to compute the same input data. As might be interpreted from the graph on Figure 10, the accelerated version consumes always more on-chip power than the sw-only version. This is explained by the additional hardware required by the accelerator. Also following the same logical thinking, stream architectures, as more resource hungry than the iterative ones, have a higher overall on-chip power consumption. Consequently, achieving superior speedups than the iterative architecture, as shown in Section VII-C. The use of a FFT accelerator translates into a maximum increase of 5mW

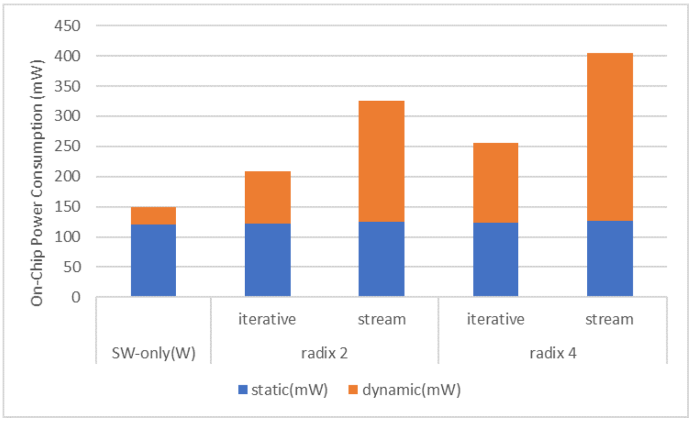

Fig. 10. Pulpino with FFT accelerator, dynamic and static on-chip power consumption at 40MHz.

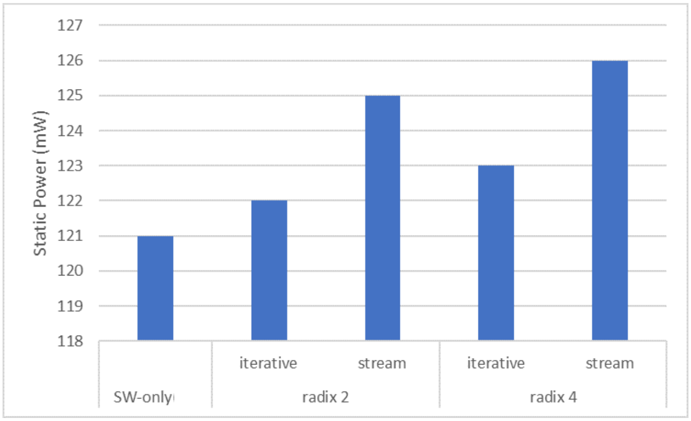

Fig. 11. Pulpino with FFT accelerator, static on-chip power consumption at 40MHz.

of static power, which corresponds to 4% of the total on-chip static power of PULPino without accelerators.

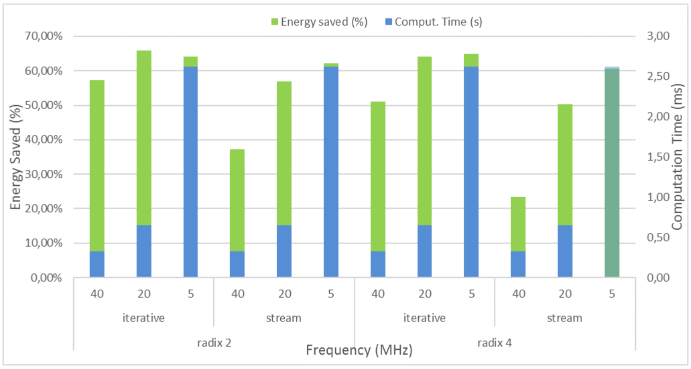

Fig. 12. Pulpino with FFT accelerator, static on-chip power consumption at 40MHz.

Figure 12 presents the power saved in percentage by using the fft accelerator, when computing an FFT algorithm. Achieving a maximum saving of 66 % when using a radix 2 iterative architecture. The ”optimal” mode of operation is achieved when the computation time is minimum and the energy savings are maximum. Thus, if a simple ratio between both of these results is calculated for every column of the graphs, is possible to point out which one would be it. Therefore, $\frac{EnergySaved}{ComputationTime} = 1.75$ when using radix 2 iterative at 40MHz, presents itself as the highest ratio among all. This might not be the best configuration for all embedded applications. Each one has its own energy restrictions and computation time requirements.

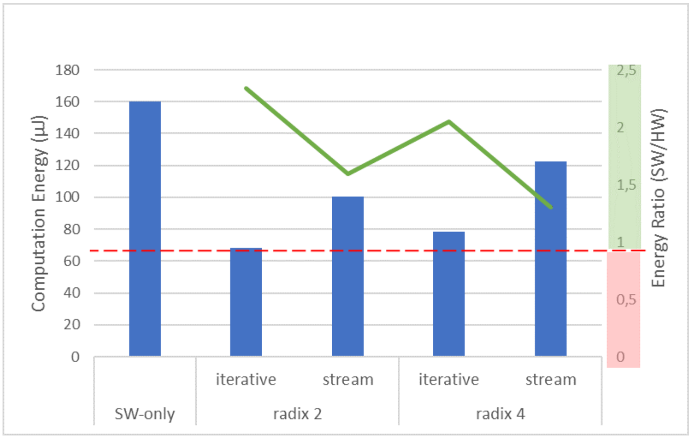

Fig. 13. Pulpino with FFT accelerator, computation energy vs energy ratio (SW/HW) at 40MHz.

On Figure 13, is portrayed a graphs that combines computation energy and energy ratio, which were calculated from the same results used on the previous graphs, using the same data input at 40MHz of main clock frequency. The computation energy figures also confirm that the FFT accelerator's architecture which saves more power is the radix 2 iterative. This can be stated by analyzing the energy ratio line. Which corresponds to the ratio between the energy consumption (on the sames graph) of both software-only and hardware accelerated versions.

VIII. Conclusions and Future Work

In conclusion, the initial goal of boosting the energy efficiency of PULPino for applications on embedded IoT devices, operating within restricted power envelopes, was successfully accomplished. The improvements were achieved by attaching two different hardware accelerators, namely a cryptographic SHA-3 accelerator and a digital signal processing FFT accelerator.

In order to successfully attach and deal with the heterogeneity between accelerator and processor, a custom low-power AXI-lite based interface was developed. Having the advantage of providing an simple and plug-n-play manner for the current and future accelerators to interface with the processor. Encouraging the development of new attachable accelerators by the open source community, since PULPino was release under an open source license. Consequently, saving development time due to reutilization of hardware designs. Paving the way for more modular embedded systems, in which is possible to add the most suitable accelerator for a certain kind of final application.
Under the scope of this Thesis, the two attached accelerators were tested on its speedup and energy efficiency. Achieving

a speedup of 185 times on the SHA-3 algorithm and 3 times on the FFT one. Being possible to achieve higher values of speedup as the input data increases, as shown by the presented tendency lines. As stated, the the cryptographic algorithm presents itself as the most suitable for acceleration in comparison to the FFT one. This can be explained by the low percentage of compressed instructions that the SHA-3 non accelerated algorithm translates into. Having only 26% of compressed instruction against 87% on the FFT non accelerated algorithm. RISC-V compressed instructions claim to reduce code-size, while enhancing energy-efficiency and performance [30].
Regarding energy savings, the SHA-3 and FFT accelerators can save up to 99.39% and 66% of energy, respectively. For the FFT accelerator, an "optimal" point of operation was proposed, setting it with a radix 2 iterative configuration at the maximum attainable frequency of 40MHz. In which among the several tested configurations, the best ratio was obtained between energy savings and the amount of time required to compute the FFT algorithm. Other modes of operation might be best suited, depending on the energy requirements of the target application.

Regarding future work, there are always room for improvements on the current AXI-lite interface, which connects the accelerator to the main AXI interconnect bus. Apart from AXI-lite, there are other kind of buses that could improve data communication between the accelerator and the AXI interconnect bus. Such as, AXI Stream, that has advantages on data streaming, but lacks individual control registers. On the previous situations, the processor is the one to fetch the required data from the memories into the accelerator, over the AXI bus. Although, higher data interchange could be achieved by featuring the accelerator with an Direct Memory Access (DMA) functionality. Allowing a direct access to such data directly from the memory. Another possible improvements for future work, is to enable the control signals received from the accelerator, to trigger the intrinsic PULPino interruptions. Meaning that the processor would not have to operate in pooling mode, having to continuously read the state of variable to be monitored, checking if it has changed. With interrupts, when the external signal is received, a flag is triggered and a proper interrupt service routine executed. As PULPino already provides such feature on its peripherals, a similar implementation could be developed for the new attachable accelerators.

References

[1] *PULP - An Open Parallel Ultra-Low-Power Processing-Platform*, 2016.

[2] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, P. Flatresse, and L. Benini. Pulp: A parallel ultra-low-power platform for next generation iot applications. In *HOTCHIPS*, 2015.

[3] Davide Rossi, Antonio Pullini, Igor Loi, Michael Gautschi, Frank K. Gürkaynak, Andrea Bartolini, Philippe Flatresse, and Luca Benini. A 60 gops/w, -1.8 v to 0.9 v body bias ulp cluster in 28 nm utbb fd-soi technology. *Solid-State Electronics*, 117:170–184, 2015.

[4] Davide Rossi, Antonio Pullini, Igor Loi, Michael Gautschi, Frank K. Gürkaynak, Jeremy Constantin, Andrea Bartolini, Ivan Miro-Panades, Edith Beignè, Fabien Clermidy, Fady Abouzeid, Philippe Flatresse, and Luca Benini. 193 mops/mw @ 162 mops, 0.32v to 1.15v voltage range multi-core accelerator for energy efficient parallel and sequential digital processing. *Cool Chips XIX*, pages 1–3, 2016.

[5] Antonio Pullini, Francesco Conti, Davide Rossi, Igor Loi, Michael Gautschi, and Luca Benini. A heterogeneous multi-core system-on-chip for energy efficient brain inspired vision. *ISCAS*, pages 2–4, 2016.

[6] M. Gautschi, P. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. Gurkaynak, and L. Benini. A near-threshold risc-v core with dsp extensions for scalable iot endpoint devices. *IEEE Transactions on VLSI Systems*, 2016.

[7] D. Rossi, I. Loi, F. Conti, G. Tagliavini, A. Pullini, and A. Marongiu. Energy efficient parallel computing on the pulp platform with support for openmp. In *2014 IEEE 28th Convention of Electrical Electronics Engineers in Israel (IEEEI)*, pages 1–5, Dec 2014.

[8] Francesco Conti, Davide Rossi, Antonio Pullini, Igor Loi, and Luca Benini. Energy-efficient vision on the PULP platform for ultra-low power parallel computing. In *Proceedings of the 2014 IEEE Workshop on Signal Processing Systems*, Piscataway, NJ, 2014. IEEE.

[9] M. Gautschi, M. Schaffner, F. K. Gürkaynak, and L. Benini. 4.6 a 65nm cmos 6.4-to-29.2pj/flop@0.8v shared logarithmic floating point unit for acceleration of nonlinear function kernels in a tightly coupled processor cluster. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 82–83, Jan 2016.

[10] F. Conti, D. Palossi, A. Marongiu, D. Rossi, and L. Benini. Enabling the heterogeneous accelerator model on ultra-low power microcontroller platforms. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1201–1206, March 2016.

[11] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, San Francisco, CA, USA, 5th edition, 2011.

[12] Brad Benton. Ccix, gen-z, opencapi: Overview & comparison. In *OPENFABRICS ALLIANCE*, 2017.

[13] Gen-Z-Consortium. *Gen-Z Overview*, 2016.

[14] Y.S. Shao and D. Brooks. *Research Infrastructures for Hardware Accelerators*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2015.

[15] Colin Schmidt and Adam Izraelevitz. A fast parameterized sha3 accelerator. Technical Report UCB/EECS-2015-204, EECS Department, University of California, Berkeley, Oct 2015.

[16] Francesco Conti, Davide Rossi, Antonio Pullini, Igor Loi, and Luca Benini. Pulp: A ultra-low power parallel accelerator for energy-efficient and flexible embedded vision. *Journal of Signal Processing Systems*, 84(3):339–354, 2016.

[17] M. Rusci, D. Rossi, M. Lecca, M. Gottardi, L. Benini, and E. Farella. Energy-efficient design of an always-on smart visual trigger. In *2016 IEEE International Smart Cities Conference (ISC2)*, pages 1–6, Sept 2016.

[18] F. Conti, D. Palossi, R. Andri, M. Magno, and L. Benini. Accelerated visual context classification on a low-power smartwatch. *IEEE Transactions on Human-Machine Systems*, 47(1):19–30, Feb 2017.

[19] Xilinx. *Fast Fourier Transform v9.0 - LogiCORE IP Product Guide*, 2015.

[20] Intel. *FFT IP Core - User Guide*, 2017.

[21] Peter A. Milder, Franz Franchetti, James C. Hoe, and Markus Püschel. Computer generation of hardware for linear digital signal processing transforms. *ACM Transactions on Design Automation of Electronic Systems*, 17(2), 2012.

[22] Andreas Traber and Michael Gautschi. *PULPino: Datasheet*, 2016.

[23] Homer Hsing. *SHA3 Core Specification*, 2013.

[24] *DFT/FFT IP Core Generator*, 2017.

[25] Peter A. Milder, Franz Franchetti, James C. Hoe, and Markus Püschel. Hardware implementation of the discrete Fourier transform with non-power-of-two problem size. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2010.

[26] Markus Püschel, Peter A. Milder, and James C. Hoe. Permuting streaming data using rams. *Journal of the ACM*, 56(2):10:1–10:34, 2009.

[27] *PULPino's Github online repository*, 2016.

[28] *A baseline Keccak implementation*, 2011.

[29] *A Simple and Efficient FFT Implementation in C++*, 2017.

[30] Andrew Waterman. Improving energy efficiency and reducing code size with risc-v compressed. Master's thesis, EECS Department, University of California, Berkeley, May 2011.

[31] Rui Policarpo Duarte and Christos-Savvas Bouganis. Arc 2014 over-clocking klt designs on fpgas under process, voltage, and temperature variation. *ACM Trans. Reconfigurable Technol. Syst.*, 9(1):7:1–7:17, November 2015.