384.178 SoC Design Lab
Integration of an Encryption Accelerator into an Open-Source Low-Power Microcontroller

Severin Jäger, 01613004

March 8, 2022

Contents

1 Introduction

Extreme edge devices are increasingly facing signal processing workloads. Sensor signals are frequently processed directly in the sensor node before being transmitted for further analysis. Examples include simple machine learning algorithms detecting relevant sequences or events in a sensor signal. However, such edge devices are usually battery powered and designed for a long lifetime, thus the energy consumption is highly constrained. So, all applications have to be designed with energy efficiency in mind. This is also a main factor when partitioning between hardware and software.

One major academic project focussing on energy-efficient processor and system design is the PULP platform [1] by ETH Zurich and University of Bologna. Over the last years, the developed several processor cores as well as peripherals and full systems on chips (SoCs). Utilising the open source instruction set architecture RISC-V, the RTL code of the IP cores developed within this project is available publicly with a permissive license.

In certain applications including biomedical devices, video surveillance, or home automation, traditional requirements are complemented by a need for confidentiality. Thus, standard cryptographic algorithms have to be executed on the hardware. Unfortunately, processing these algorithms in software is relatively complex and consumes significant memory, time, and thus energy. These issues might however be alleviated by offloading the computation to a dedicated cryptographic accelerator. The scope of this project is integrating and open-source hardware accelerator into the PULPissimo SoC [2] and comparing it to a pure software solution.

This report is structured as follows. Firstly, Section 2 discussed the relevant background including the PULPissimo SoC and AES hardware. Then, Section 3 elaborates on the work conducted during this project. Section 4 outlines the key results before Section 5 concludes this report.

2 Background

2.1 PULPissimo

PULPissimo is a microcontroller and as such one of the smallest platforms within the PULP ecosystem. It comes with a single RISC-V core, memory, a DMA controller and a full set of peripherals connected to a peripheral bus. Furthermore, it allows the convenient integration of a so-called hardware processing engine (HWPE) which directly operates on the memory. The SoC architecture is depicted in section 2.1.

The design is mainly targeting ASIC technology and focussed on maximum energy efficiency. In a 22 nm FDX technology, an energy efficiency of 433 MOPS/mW has been shown [2]. The full System Verilog RTL code is available on GitHub [3].

Hardware Processing Engine (HWPE)

The PULP hardware processing engine is a quite unique hardware accelerator. As depicted in Section 2.1, it consists of a data interface, a control interface and the data path. In contrast to most accelerators in literature it neither uses a dedicated accelerator interface in the CPU nor direct memory access (DMA) to get the data to process. Instead, it

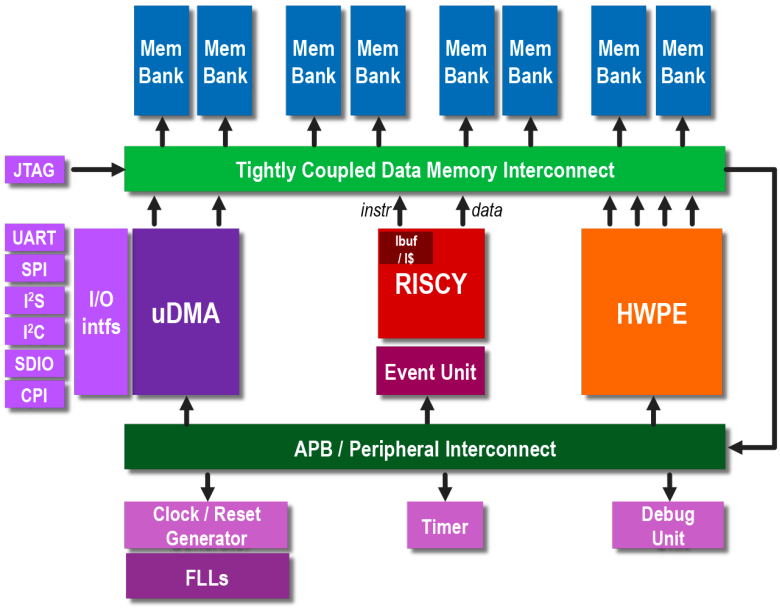

Figure 1: Architecture of the PULPissimo SoC [3]

shares the so-called tightly coupled memory (which somehow corresponds to an L2 cache) with the processor core. The control interface is connected to the peripheral bus and is thus seen as a memory-mapped peripheral by the CPU. It provides configuration registers which can be written by the processor.

A typical HWPE operation works as follows. Firstly, the relevant data has to be in the shared memory. Then, the CPU sets all relevant configuration data (e.g. pointers of source and destination in shared memory, HWPE configuration) in the HWPE via the control interface. Once an HWPE operation is started, it operates fully autonomous and the CPU can perform other computations. After completion, the HWPE trigger and event which can be polled in the CPU or raise an interrupt. The event indicates that the output data is written to the shared memory.

HWPEs are not limited to PULPissimo, but can also be integrated into more powerful SoCs within the PULP project including multicore systems and HPC clusters. Thus, the effort of designing an HWPE can be reused easily. Furthermore, a template HWPE implementing a multiply-accumulate (MAC) engine is available [4]. This template features a relatively simple data path, but a powerful control interface including a microcode processor and a finite state machine as well as three 32 bit inputs and one 32 bit output. The template is intended as starting point for new HWPEs and alleviates the burden of the developer to write a device driver and to handle the memory access.

2.2 AES Hardware

The advanced encryption standard (AES) defines a set of block cipher algorithms which are also known under their original name Rijndael. It works on 128 bit words and uses either 128, 192, or 256 bit long keys. The algorithm was designed for straightforward hardware implementation and is one of the most used block ciphers.

Thus, several hardware implementation of the AES algorithms are available. As this

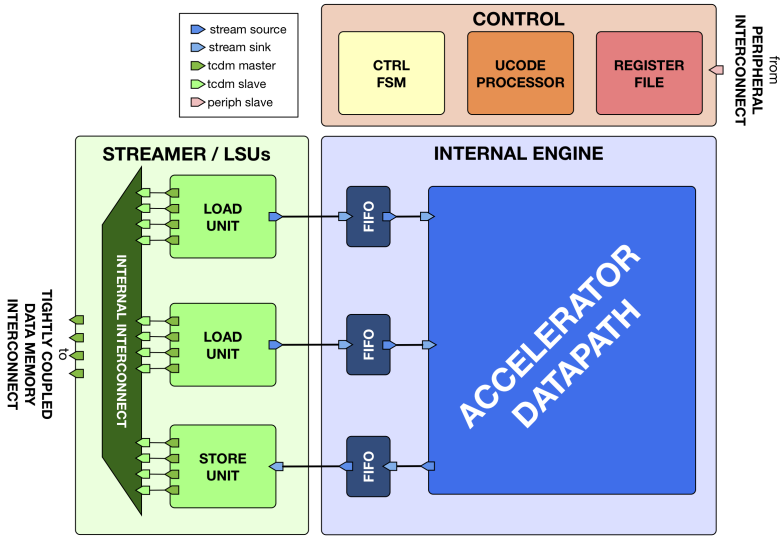

Figure 2: Architecture of a PULP Hardware Processing Engine [5]

project uses open-source System Verilog RTL code, only open-source Verilog implementations were considered. Section 2.2 compares the three IP cores considered. An edge device requires only encryption capabilities to transfer sensor data in a secure way. Also due to size and energy limitations, only AES with 128 bit keys was considered.

Core	`tiny_aes` [6]	`aes_128` [7]	`secworks_aes` [8]
Cycles/Op	1	12	4
Latency (cylces)	21	12	14
LUTs	4588	487	3327
Registers	4474	402	2990
BRAM Tiles	68	5	0
Max. Frequency [MHz]	375.9	180.5	124.8
Decryption	no	no	yes
256 bit keys	no	no	yes

Table 1: Comparison of selected open-source AES IP cores. Resource consumption and clock frequency were obtained with Vivado 2020.2 synthesising for a Nexys4DDR evaluation board.

Clearly, the `tiny_aes` core provides the best performance. It is fully pipelined and thus has the highest throughput and the shortest critical path. In contrast, `aes_128` is a highly minimal design that reuses the stages of the AES algorithm to minimise hardware resources. In between, the `secworks_aes` core finds a trade-off and optionally offers several features that are not required for this project (decryption or larger keys). Given the energy constraints and the most likely relatively low number of transmissions required, the `aes_128` IP core was selected to be integrated into PULPissimo as HWPE.

3 Implementation

As most of the work done during this project is based on existing hardware and software, this section describes the required implementation steps.

3.1 Setup

As PULPissimo is written in System Verilog and the scripts require either Mentor Questasim or Cadence Xcelium, the ICT EDA server was used to run the Mentor toolchain. This required working on the server file system and mounting the relevant directories via ssh on the local machine. As the PULPissimo scripts make use of environment variables frequently, doing so required several scripts, some of the are located in the `scripts` directory in this repository.

3.2 Reference Software

To have a reasonable baseline, a software implementation of the AES algorithm was required. Due to its small code size which perfectly fits microcontroller environments, the Tiny AES in C [9] implementation was chosen. It easily compiles for PULPissimo with the PULP SDK and requires only around 2 kiB of memory. The software library offers several encryption modes and key sizes, due to the limited requirements for this project only the electronic code book (ECB) encryption capability with 128 bit keys is required.

In the simple demo application in the `aes_sw` directory, a single 128 bit word can be encrypted N times to eventually compare it to the HPWE runtime. Basic verification is also in place comparing the result with the known encryption result (this is trivial for ECB encryption).

3.3 Hardware Processing Engine

The design of the AES HWPE started with the MAC engine template [4]. As a first step, the data path was removed and replaced by a simple feed-through behaviour to implement data mover. Then, the AES IP was added and two of the inputs provided 32 bit word and key inputs. The third input was disabled and thus removed by the simulation and synthesis tools. As a next step, two simple IP blocks that combine four 32 bit words into one 128 bit words and vice versa were designed to provide 128 bit inputs and utilise the 128 bit output of the AES core. For both blocks, a simple verification setup was created.

The resulting AES HWPE is depicted in Section 3.3. The input and output streamers are controlled by the FSM in the control block which uses the registers written by the CPU beforehand. All data path units use a simple ready/valid handshake to allow both forward and backward dependencies, e.g. if the AES unit is busy of no new data is provided by the input streamers yet. The registers were used to balance the pipeline, however no timing analysis has been undertaken to examine whether they are necessary eventually.

Additionally, a simple demo application comparable to the reference software demo was implemented. It writes input word and keys to the memory (N times), sets all required control registers via the HWPE driver, and triggers N encryptions. In the end, the results are again compared to their expected values.

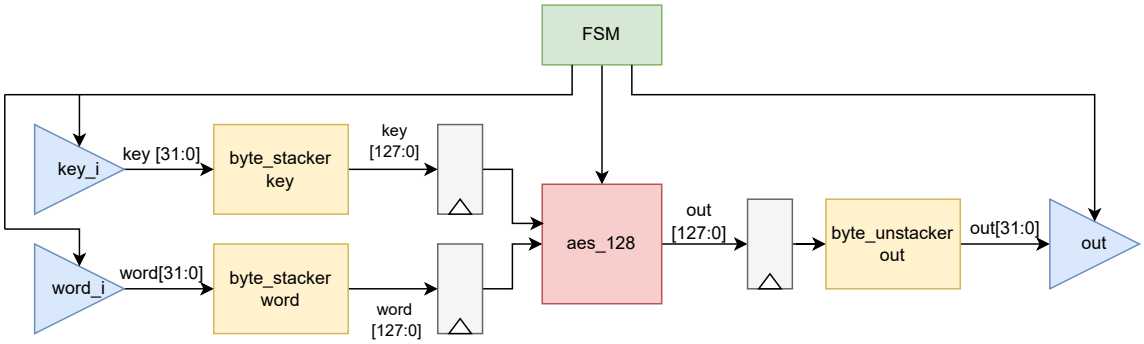

Figure 3: Block diagram of the implemented AES HWPE

4 Results

The key results achieved are summarised in Section 4. Clearly, the hardware implementation outperforms the software implementation in terms of runtime. Also the code size is notably reduced (most of the code size comes from the runtime code). The speed-up is likely do directly translate into energy savings as the CPU can enter a sleep mode after encryption more quickly. Unfortunately, issues with the synthesis scripting prevented area estimations. Still, the `aes_128` IP core is very small (s. Section 2.2), thus the area overhead relative to the PULPissimo SoC should be minimal.

	AES Software	AES HWPE
Runtime $N = 32$	2876 μs	10 μs
Relative speed-up	1	287.6
Runtime further encryption	61 μs	280 ns
Code size	9816 bytes	9140 bytes

Table 2: Simulation results for both AES implementations. The simulation clock frequency is 50 MHz and the used engine was Mentor Questasim.

5 Summary and Outlook

During this project, a PULP hardware processing engine implementing AES encryption was designed and integrated into the PULPissimo SoC. As a result, an encryption speed-up of 287.6 was reached while decreasing the code size. However, this projects motivates further work. It might be interesting to support also more complex (but in reality inevitable) encryption modes like CBC. Furthermore it might be interesting to use modern lightweight cryptographic algorithms designed specifically for edge devices.

Besides that, an ASIC implementation of the AES HWPE could bring deeper insights into both area and power consumption. This also matches the PULPissimo philosophy which is using FPGA only as prototyping platform and is intended to be implemented as IC.

References

[1] *Pulp platform website*, https://www.pulp-platform.org/, Accessed March 7, 2022.

[2] P. D. Schiavone, D. Rossi, A. Pullini, A. D. Mauro, F. Conti, and L. Benini, "Quentin: An ultra-low-power PULPissimo SoC in 22nm FDX," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, IEEE, Oct. 2018.

[3] *PULPissimo repository*, https://github.com/pulp-platform/pulpissimo, Accessed March 7, 2022.

[4] *Hardware MAC engine repository*, https://github.com/pulp-platform/hwpe-mac-engine, Accessed March 7, 2022.

[5] *Hardware processing engines - interface specification*, https://hwpe-doc.readthedocs.io/en/latest/protocols.html, Accessed March 7, 2022.

[6] *Tiny_aes IP core*, https://opencores.org/projects/tiny_aes, Accessed March 7, 2022.

[7] *Aes_128 IP core*, https://github.com/www-asics-ws/aes_128, Accessed March 7, 2022.

[8] *Secworks AES IP core*, https://github.com/secworks/aes, Accessed March 7, 2022.

[9] *Tiny AES in C repository*, https://github.com/kokke/tiny-AES-c, Accessed March 7, 2022.