Hardware Acceleration of HDR-Image Tone Mapping on an FPGA-CPU platform through High-Level Synthesis

Mattia Cacciotti, Vincent Camus, Jeremy Schlachter, Alessandro Pezzotta, Christian Enz
Integrated Circuits Laboratory (ICLAB)
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
mattia.cacciotti@epfl.ch

***Abstract*—In this paper, the hardware acceleration of a tone-mapping algorithm for High-Dynamic-Range image processing is presented. Starting from the C++ source code, High-Level Synthesis has been performed using Xilinx SDSoC for a Xilinx Zynq SoC device. After an initial code optimization to improve the memory access bottleneck, SDSoC pragmas have been introduced to boost system performance through an increased parallelism. Preliminary results have shown significant reductions in the execution time and the energy consumption compared to the conventional software implementation.**

***Index Terms*—Heterogeneous systems, FPGA, high-level synthesis, hardware-software co-design, image tone mapping.**

I. Introduction

By 2025, 1 trillion devices will be connected to the internet [1], sending an incredibly large amount of data to cloud servers, which already consume more than 2 % of the world's produced electricity [2]. With the breakdown of Dennard's scaling, energy issues are becoming dominant to the development of future computing platforms, both at cloud server scale and IoT edge. At the sensor node, low-power techniques such as approximate computing [3] have been developed to trade off computation exactness for lower power consumption and increased battery life. While at the high-performance and cloud-computing scale, specialized hardware accelerators such as Field Programmable Gate Arrays (FPGA) have gained considerable interest thanks to their capability in performing massive amounts of operations in parallel with a lower energy cost [4].

The increasing demand for custom hardware has driven embedded software developers towards more complex architectures, employing heterogeneous systems which combine together processing units and hardware accelerators [5]–[7]. By doing so, computationally-intensive operations can be unloaded from the CPU to the dedicated circuit.

Developing application-specific hardware accelerators for sophisticated applications, however, is not trivial and requires complex coding in specialized Hardware Description Languages like VHDL or Verilog. To provide a solution to this problem, High-Level Synthesis (HLS) tools have been developed to directly map a C/C++ algorithm into a digital circuit. Even if a faster and more efficient design-space exploration is allowed by HLS, these tools are not yet independent and a solid knowledge of hardware design is still required to interpret results properly and guide the optimization efficiently.

This work aims at presenting the hardware acceleration of a popular image processing algorithm, the tone mapping, on a Xilinx Zynq SoC, a heterogeneous platform which combines an ARM-based processing unit together with programmable logic. Xilinx SDSoC has been used as development environment and it allows to perform hardware-software co-design through HLS and a C/C++ compiler respectively.

The paper is organized as follows: Section II gives a general overview of tone mapping and focuses on its algorithmic implementation; Section III presents the target platform and design tool and describes the developed optimization flow for the hardware acceleration; Section IV shows the experimental results obtained for execution time and energy consumption.

II. HDR-Image Tone Mapping

Tone-mapping is an image processing algorithm that has recently gained more and more importance due to the spreading of High-Dynamic-Range (HDR) images in mobile phones and other portable devices. Those images are characterized by a very high ratio between the luminance of the brightest and the darkest pixel and are able to represent a wide range of luminance values.

Tone mapping is widely used to match the dynamic range of the captured image with the one of the device where it is going to be displayed, which is usually limited. This algorithm essentially allows to optimize the contrast, which is directly related to the dynamic range, while still preserving the high resolution of the image and its color appearance. In this way, dark zones will become brighter while bright zones will become darker.

Different tone-mapping techniques are available, but the algorithms can be overall classified in two groups: global and local. In the former, the same transformation applies to all the pixels of the image, while in the latter each pixel transformation depends on the value of its neighboring pixels.

This work was funded by the Swiss-French SNSF/ANR ARTEFaCT project.

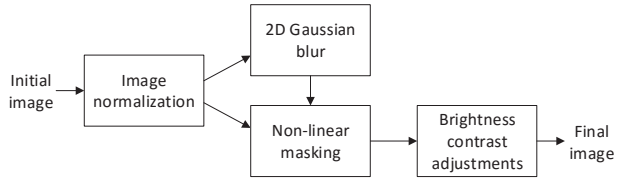


Fig. 1: HDR-Image Tone Mapping block diagram [8].

A. Algorithm implementation

The algorithm considered in this work belongs to the second group [8, 9] and it has been implemented as a C/C++ code. Fig. 1 shows its block diagram. The input HDR image goes through the following steps:

1) Image normalization: each pixel inside the input image is normalized with respect to their maximum value.
2) Gaussian blur: bi-dimensional image filter in which each pixel is updated summing up to it a certain number of adjacent pixels, horizontal or vertical, weighted by a certain coefficient. The number of adjacent pixels and the weights of the multiplications are determined by width and magnitude of a Gaussian distribution.
3) Non-linear masking: main tone mapping operation used to modify through gamma-correction the pixel values of the original image using the pixels of the blurred image.
4) Brightness and contrast adjustments to improve quality.

III. Hardware-Software Co-Design

A. Target platform and design tool

The platform targeted for the design implementation has been a Xilinx Zynq-7000 All Programmable System-on-Chip (AP SoC) [10], a heterogeneous system that combines the flexibility of programmable logic together with the software programmability of an ARM-based processor.

Given the nature of the Zynq SoC, Xilinx SDSoC (Software Defined SoC) [11] has been used to perform hardware-software co-design. This recent tool suite is an Integrated Development Environment (IDE) dedicated to Zynq SoC devices that allows to generate both the ARM software and the FPGA bitstream. For the hardware part, the SDSoC compiler invokes Xilinx Vivado HLS to compile synthesizable C/C++ functions into programmable logic. For the software part, instead, the accelerated functions are replaced by software stubs and the code is compiled and linked using a standard GNU toolchain. The SDSoC design flow is presented in Fig. 2.

Given a specific application running on ARM, the code is profiled to determine the most computationally-intensive functions. Once identified, these functions are selected for hardware acceleration in order to be off-loaded from the CPU execution. However, this operation might not guarantee an immediate improvement in system performance, especially when memory accesses are not optimized. CPUs, in fact, have usually faster random accesses to external memories than programmable logic, thanks to caches and higher clock frequencies. Hence, algorithm restructuring might be necessary

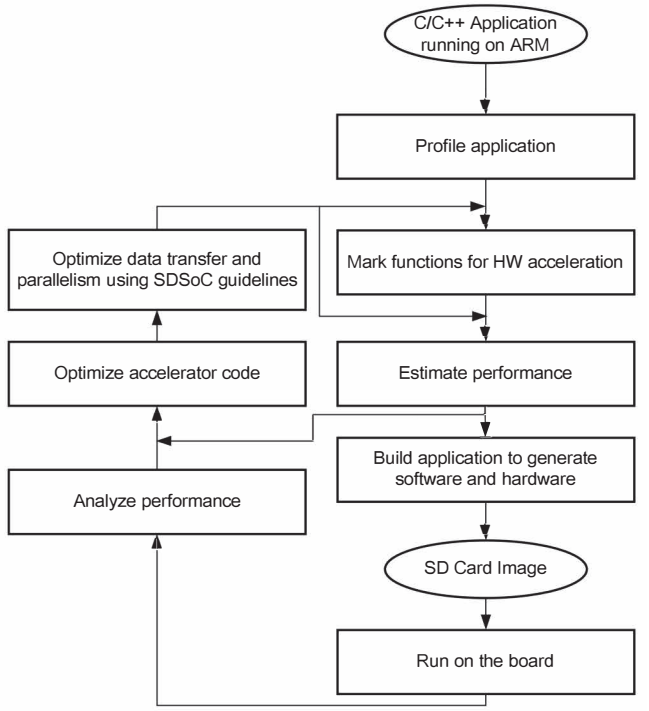


Fig. 2: SDSoC design flow [11].

to transform random memory accesses into sequential accesses and, if dependencies make it unfeasible, to exploit local data buffers using memory blocks inside the FPGA (Fig. 3).

Only once the data transfer bottleneck is solved, it will be possible to obtain significant benefits from the computational parallelism offered by the FPGA architecture.

B. Algorithm optimization

Given the aforementioned considerations, the tone-mapping algorithm has been profiled and the Gaussian blur function identified as the most computationally-intensive. Since the code was not optimized for hardware acceleration, the first system implementation using SDSoC did not show any improvement in the execution time. An extensive amount of random memory accesses, in fact, was required in the marked function, since neighboring pixels (preceding and succeeding) were needed to perform blurring.

Therefore, the data flow has been restructured to make the accesses to the external memory sequential (Fig. 4). Pixels are now sequentially read from the off-chip RAM and stored in a local buffer inside the programmable logic, the block RAM (BRAM). Once the buffer becomes full, the

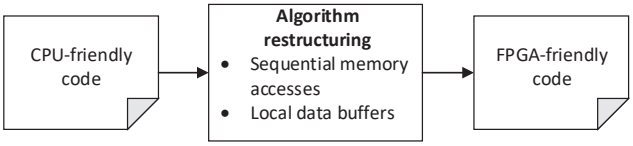


Fig. 3: Algorithm restructuring.

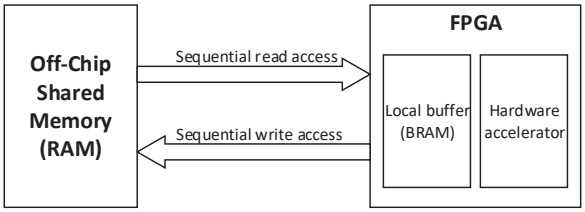


Fig. 4: FPGA sequential accesses to external memory.

Gaussian blur starts the computation and each new streamed pixel substitutes the oldest one in the buffer. Each output pixel is then sequentially written back to the external memory.

The advantage of HLS does not only lie in the possibility to accelerate functions in hardware avoiding coding complex Verilog or VHDL, but also to have a faster and more efficient design space exploration [12]. Compiler directives called *pragmas* can be used in SDSoC to guide the compilation to improve system performance at a higher abstraction level, essentially controlling the following knobs:

- Data motion network, to specify both the most suitable data mover between software routine and hardware function and the kind of access pattern employed (i.e. random or sequential).
- System parallelism, to improve the parallelism between loop iterations through loop pipelining or loop unrolling and to increase the memory bandwidth by local memory blocks reshaping.

Different implementations of the Gaussian blur function have been synthesized to decrease the execution time. At each optimization step, the performance report obtained after the compilation has been analyzed to identify the bottleneck of the design. This report shows for each clock cycle which operation is performed by the hardware module. The pragmas that have been added to the source code to boost the performance are:

- *#pragma HLS PIPELINE*, used to increase the parallelism of the loops required for pixel processing, instantiating in hardware multiple iterations of these loops (which would else be executed sequentially in software). Vivado HLS performs this operation trying to minimize the initiation interval, i.e. the number of clock cycles necessary between consecutive loop iterations. Data dependency and hardware resources might limit this optimization.
- *#pragma HLS ARRAY_PARTITION*, used to map and partition software-defined arrays into specific FPGA memory units (e.g. BRAMs or registers). In this way, the throughput of read/write operations is increased thanks to the availability of a higher number of memory ports.

C. Fixed-point conversion

After having optimized the system at the architectural level, the focus has been moved towards the successive bottleneck, the arithmetic. Despite floating-point representation is a highly accurate data type simple to use at the software level, it moves all the complexity down to hardware in complex Floating-Point Units (FPUs). For this reason, most low-power embedded systems prefer to adopt fixed-point arithmetics [13]. This type of representation shifts the complexity up to software, allowing the use of simple hardware operators implementing integer arithmetic and improving speed, area and energy at the price of a reduced, but acceptable, accuracy.

Therefore, the Gaussian blur function has been converted from floating-point to fixed-point to further reduce the execution time while still keeping accuracy within reasonable boundaries. To this extent, the Vivado HLS C++ arbitrary precision data type *ap_fixed* has been used, choosing 16-bit as total number of bits. When designing through SDSoC, not all arbitrary bit widths can be chosen for the arguments of hardware functions, due to communication issues between FPGA and processing system. In order to guarantee the bus alignment, in fact, the width must be 8, 16, 32, or 64 bits.

IV. Experimental results

The experimental results shown in this section have been obtained providing as input to the system the HDR image shown in Fig. 5a, with a size of 1024×1024 pixels.

A. Execution time

In order to provide an execution time reference, the C++ source code has been fully executed on the ARM processor embedded in the Zynq SoC. Then, after having marked the Gaussian blur function for acceleration in the programmable logic, the design has been optimized following the steps listed in Table I.

TABLE I: Hardware acceleration optimization steps.

1	Algorithm restructuring for sequential memory accesses
2	Pipelining and array partitioning through HLS pragmas
3	Floating-point to fixed-point conversion

The tone mapping execution times for the different implementations are presented in Table II. The table entry *Marked HW function* shows how a straightforward selection of the most computationally-intensive function for hardware acceleration would not produce any immediate gain. Most likely, it will lead to a degradation of the performance due to a poor compatibility of a general software code with the FPGA computing paradigm.

Execution times are summarized in Fig. 6, omitting the *Marked HW function* which is not relevant. The bar chart underlines both the time spent in the programmable logic (PL)

TABLE II: Tone mapping execution times.

Design implementation	**Execution time**	
	Gaussian blur	**Total**
SW source code	7.29 s	26.66 s
Marked HW function	176.00 s	195.28 s
Sequential memory accesses	17.02 s	35.34 s
HLS pragmas	0.79 s	19.10 s
FlP to FxP conversion	0.42 s	19.27 s

(a) Initial HDR image (b) 32-bit Floating-Point (c) 16-bit Fixed-Point

Fig. 5: Initial 1024×1024 HDR image (a) and tone-mapped versions obtained using either floating-point (b) or fixed-point (c) Gaussian blur accelerators.

for the execution of the Gaussian blur and the one spent in the processing system (PS) for the rest of the algorithm. Overall, an execution time improvement of more than 17× has been achieved for the final hardware accelerated Gaussian blur with respect to its original software version. The optimization steps that contributed the most to the speed-up have been the algorithm restructuring and the pipelining and array partitioning.

B. Image quality evaluation

The tone-mapped images resulting from *HLS pragmas* and *FlP to FxP conversion* implementations are shown in Figs. 5b and c, respectively. To evaluate the image quality degradation, the Peak Signal-to-Noise Ratio (PSNR) has been computed for the 16-bit FxP version taking the 32-bit FlP image as a reference.

The resulting PSNR is equal to 66 dB, which is similar to the typical values obtained in lossy image compression [14]. However, no real visual difference between the two images can be noticed. To this extent, a better idea of image quality degradation is given by the Structural Similarity (SSIM) index [15]. This quality metric measures the similarity between two images, hence it is somehow closer to human perception. As a result, when computed between the floating-point and the fixed-point implementations, the resulting SSIM is equal to 1, which corresponds to the same image quality.

C. Energy consumption

Core and auxiliary voltages are provided to the Zynq SoC by Texas Instruments (TI) power controllers. These devices feature a Power Management Bus (PMBus) that can be used to communicate with an external PC through an USB-to-GPIO adapter. By using the TI Fusion Digital Power Designer GUI, is then possible to monitor the power consumption of the system. Among the ten different power rails available, the focus has been put on those powering up the main components, i.e. the programmable logic (PL), the processing system (PS) and the memories (DDR and BRAM).

Intuitively, when moving from a purely software implementation to one including a hardware accelerator, the overall power consumption increases, since the synthesizer enables a growing amount of logic circuits in the PL. On the other hand, the presence of an accelerator allows to reduce the time required to complete an operation. Hence, a more significant figure is given by the energy instead of the power.

The energy values reported in Fig. 7 have been obtained multiplying the average power consumption measured with

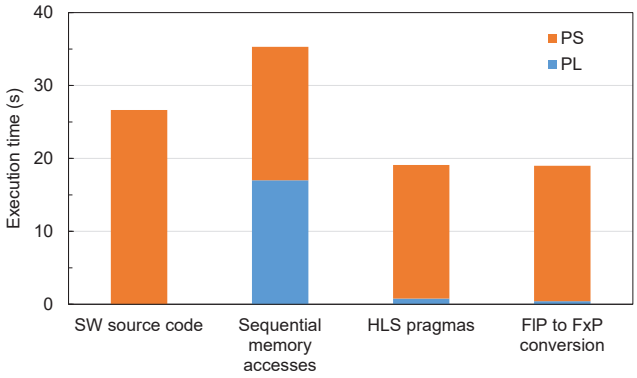


Fig. 6: Tone mapping execution time.

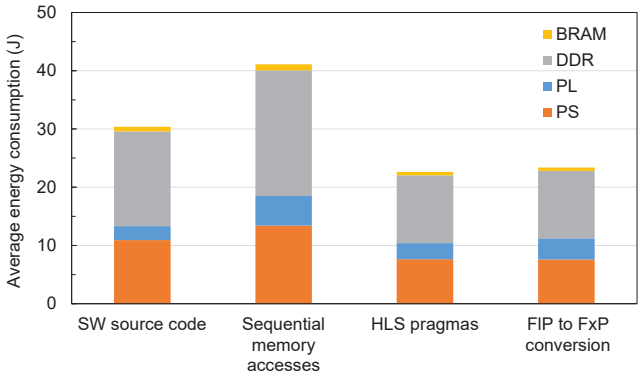
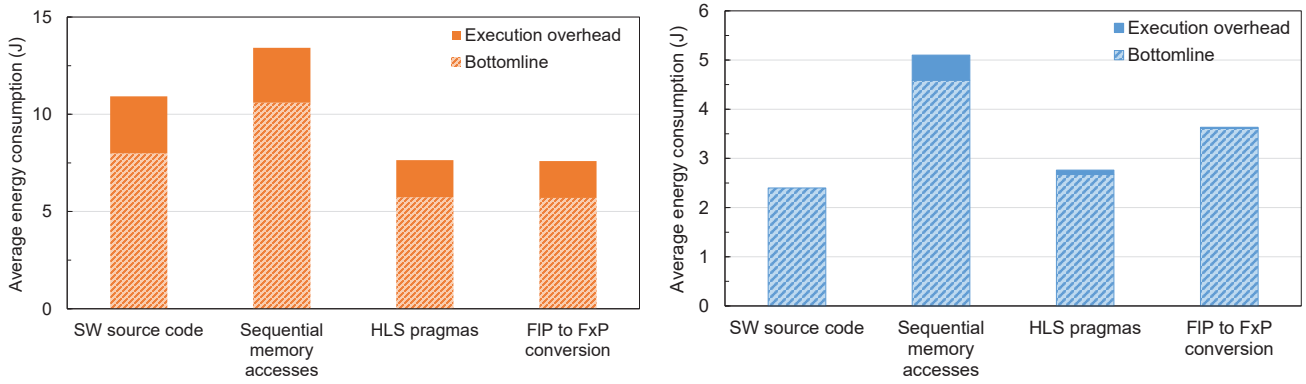


Fig. 7: Tone mapping average energy consumption.



(a) Processing System (PS)

(b) Programmable Logic (PL)

Fig. 8: Energy bottomline and execution overhead for PS (a) and PL (b).

the TI software by the corresponding execution time. Compared to the initial software-only version, the final fixed-point hardware-accelerated implementation allows a 23 % energy consumption reduction for each processed image, going from 30 J down to 23 J.

The measured energy can be divided in two contributions, namely the *bottomline* and the *execution overhead*. The first term refers to the energy consumed by the system when it is in idle state waiting for the application to be executed, while the second represents the additional energy required to perform the computations. Figs. 8a and b show this distinction for the processing system and the programmable logic, respectively.

The energy consumption for the DDR and the BRAM is not reported in this chart because it does not vary when moving from idle to execution. As expected, for the PS, shorter execution times allows to reduce both the bottomline and the execution overhead energy terms. A more interesting insight, however, is given by the PL. The bottomline term, in fact, increases when going from *SW source code* to *FlP to FxP conversion*, due to an increasing amount of programmable logic being used. On the other hand, the execution overhead (which is not present in the software-only version) decreases thanks to the very short execution times.

V. Conclusion

In this paper, the hardware-software co-design steps necessary to accelerate a HDR-image tone-mapping algorithm on a Xilinx Zynq SoC platform through Xilinx SDSoC have been explored.

Despite a faster and more efficient design-space exploration is allowed by HLS-based tools, the designer still needs to be aware of the hardware implementation constraints in order to guide the optimization flow effectively.

After having profiled the source code to identify the most computationally-intensive software routine for the embedded ARM CPU, the Gaussian blur function has been marked for hardware implementation in the programmable logic of the FPGA. Further optimizations have been performed through algorithm restructuring, loop pipelining, array partitioning and fixed-point conversion, targeting at each step the computation bottleneck. Overall, an improvement of 17× in the execution time has been achieved for the accelerated function together with a 23 % energy consumption reduction.

References

[1] J. Heinlein, in *ARM TechCon, Cambridge, UK*, Oct. 25 2016.

[2] "After Moore's law - Technology Quarterly," *The Economist*, March 12 2016. [Online]. Available: http://www.economist.com/technology-quarterly/2016-03-12/after-moores-law.

[3] V. Camus, M. Cacciotti, J. Schlachter, and C. Enz, "Design of approximate circuits by fabrication of false timing paths: The carry cut-back adder," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS)*, June 2018.

[4] A. Putnam, A. M. Caulfield *et al.*, "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services," in *Computer Architecture (ISCA), ACM/IEEE 41st International Symposium on*, 2014, pp. 13–24.

[5] L. Suriano, A. Rodriguez, K. Desnos, M. Pelcat, and E. de la Torre, "Analysis of a heterogeneous multi-core, multi-HW-accelerator-based system designed using PREESM and SDSoC," in *IEEE Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, July 2017.

[6] S. D. Roh, K. Cho, and K. S. Chung, "Implementation of an LDPC decoder on a heterogeneous FPGA-CPU platform using SDSoC," in *2016 IEEE Region 10 Conference (TENCON)*, 2016, pp. 2555–2558.

[7] S. Nouri, J. Rettkowski, D. Göhringer, and J. Nurmi, "HW/SW co-design of an IEEE 802.11a/G receiver on Xilinx Zynq SoC using high-level synthesis," in *ACM Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, June 2017.

[8] V. Camus, J. Schlachter, M. Gautschi, F. K. Gurkaynak, and C. Enz, "Approximate 32-bit floating-point unit design with 53% power-area product reduction," in *European Solid-State Circuits (ESSCIRC), IEEE 42nd Conference*, Sept 2016, pp. 465–468.

[9] N. Moroney, "Local Color Correction Using Non-Linear Masking," in *Color Imaging Conference (CIC), 8th IS&T/SID*, 2000, pp. 108–111.

[10] Xilinx Inc., *Zynq-7000 Technical Reference Manual (UG585)*.

[11] Xilinx Inc., *SDSoC Environment User Guide (UG1027)*.

[12] Xilinx Inc., *SDSoC Environment Profiling and Optimization (UG1235)*.

[13] D. Menard, D. Chillet, and O. Sentieys, "Floating-to-Fixed-Point Conversion for Digital Signal Processors," *EURASIP Journal on Advances in Signal Processing*, vol. 2006, no. 1, Jan 2006.

[14] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," *Electronics Letters*, vol. 44, no. 13, pp. 800–801, June 2008.

[15] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.