Developing Custom RISC-V ISA Extensions for General Embedded Image Processing Operations

Stephan Nolting[1], Ingo Hoyer[1], Alexander Utz[1], Holger Kappert[1] and Guenter Grau[2]

[1]Fraunhofer Institute for Microelectronic Circuits and Systems (IMS), Department for Smart Sensor Systems, Duisburg, Germany
[2]advICo microelectronics GmbH, Recklinghausen, Germany

Abstract

Processing data-intense tasks right inside an image sensor system allows to offload real-time constraints from centralized data processing nodes and also to reduce transmission bandwidth requirements. High-performant application-tailored processors emblematize one feasible concept to cope with these demands while still providing flexibility for future revisions of the actual algorithms. This work presents an area-optimized customized RISC-V processing system designed as application-specific instruction set processor (ASIP), which is optimized to accelerate exemplary image processing operations. The proposed ISA extensions exploit single-instruction multiple-data (SIMD) concepts on pixel level, resulting in a speedup of 13 (compared to a software-only approach) while increasing the prototype's FPGA hardware utilization by just 15%.

Introduction

Performing data-intense operations "on-site", i.e. right inside the data-sampling sensor system, is a common trend. For example, camera systems can extract image features, compress frame data or even compute entire scene analysis algorithm without interaction with a host. This can help reducing transmission bandwidth, but also allows to shift real-time constraints to the device itself offloading a centralized data processing system. By performing data-intense tasks close to the sensor, like image pre-processing/conditioning, time-critical parts of the algorithm can already be processed on-site. High processing throughput as well as energy efficiency are crucial - especially if those devices are battery-powered. Using application-specific hardware extensions is one option to tackle those requirements.

In this work an exemplary cut-out of a real-world image processing application is used. At first, common operations and their computational bottlenecks are identified. Based on this, dedicated accelerators are developed, which are implemented as custom instruction set architecture (ISA) extensions of a base CPU core. The proposed ISA extensions are finally benchmarked on a FPGA prototyping platform analyzing their performance speedup and hardware costs.

A generic RISC-V base CPU is used as starting point. The RISC-V instruction set architecture (ISA) itself was designed to support the option for custom ISA extensions options [1]. Exploiting these options on CPU-level allows to setup an application-specific instruction set processor (ASIP), which is specialized for the given application. This approach allows an efficient hardware-software partitioning as compute-intensive parts can be implemented as dedicated hardware extension while keeping the design flexible due to software programmability.

System Overview

The area-optimized and highly configurable and extensible open-source NEORV32 RISC-V core [2] has been selected as initial base CPU. This generic processing core already provides hardware/software templates for implementing application-defined extensions as custom instructions ("CI") and is configured as RISC-V `rv32im_Zicsr` architecture. The core is integrated into an application-specific system-on-chip (SoC) including embedded memories and further peripheral modules (illustrated in Fig. 1a). Exemplary image data is sampled as 8-bit grey-scale values via a standard serial peripheral interface (SPI) and placed into an internal frame buffer (FBUF). The firmware uses a sliding-window approach to process the input image calculating the arithmetic mean and consecutively the standard deviation for each window. Pixel data is fetched from the framebuffer in a packed form by the CPU (four 8-bit pixels per 32-bit word) allowing to exploit sub-word parallelism using a SIMD-based processing approach.

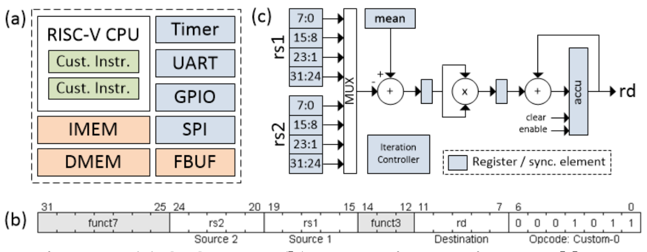

Figure 1: (a) SoC setup, (b) custom instruction word layout and (c) CI logic for the standard deviation

Custom Instructions

The RISC-V specification [1] defines several distinct opcodes being explicitly reserved for custom use. For this work, we decided on using the `custom-0` opcode (`0b0001011`) to implement custom R3-type instructions. This instruction format provides two source registers, one destination registers and two function-defining immediate field bit-fields (compare with Fig. 1b). In total, three custom instructions are implemented. Two of them use a SIMD-based processing approach to accelerate computing the arithmetic mean and the standard deviation: `SIMD_sqdsum` computes the accumulated sum of squared differences (for standard deviation) and returns the lowest 32-bit of the accumulator to the destination register `rd`. `SIMD_sum` computes the accumulated sum over all sub-words (for arithmetic mean) and also returns the lowest 32-bit of the accumulator to the destination register.

`SIMD_sqdsum`: rd <= $\text{accu} \mathrel{+}= \sum_{i=0}^{N-1}(mean - op_i)^2$
`SIMD_sum`: rd <= $\text{accu} \mathrel{+}= \sum_{i=0}^{N-1} op_i$

Both instructions can process up to eight 8-bit wide subwords (packed as `rs1` and `rs2` input operands) computing a single 32-bit accumulated result. The general architecture of the `SIMD_sqdsum` instruction, is shown in Fig. 1c. Single bits in the `func` bitfields of the custom instruction words are used to control processing steps (i.e. clearing the accumulator when starting the processing of a new window or writing data to the CI-internal `mean` register so no additional source port is required). Additionally, the standard deviation requires the computation of a square root. This function has also been implemented as custom instruction processing an unsigned 32-bit integer and resulting a 32-bit fixed-point number with 16 fractional bits. All custom instructions operate in serial manner to keep hardware requirements at a minimum. Consequently, the CI's functional units need several cycles to complete operations.

Evaluation

The processing platform is implemented and evaluated on a low-power Lattice iCE40 UltraPlus FPGA (`iCE40UP5k`) [3]. The design is clocked at 24MHz and uses the FPGA's large-scale "SPRAM" primitives for instruction and data memories. The resource requirements for implementing the base RISC-V CPU core ("CPU" row) and the proposed ISA extensions ("CI" row) are listed in Table 1. A 64x64 pixel reference image with 8-bit grey-scale data per pixel is used as reference processing input. The image is fetched via SPI from FPGA-external storage. Table 2 presents the required clock cycles for processing an entire image using a window size of 7x7 pixels. The "SW-only" column lists the clock cycles for a plain-software implementation (i.e. no CIs). Utilizing the proposed ISA extensions provides a total speedup factor of 13. The computational bottleneck is identified to be the computation of the standard deviation. This part alone provides a speedup factor of 35 when using the proposed ISA extensions ("SW+CI" column). The total frame rate is increased from 0.91 frames per second (fps) to 13.1 fps resulting in an overall speedup factor of 13.4. The hardware of all three custom instructions sums up to 530 LUTs increasing the base CPU hardware by approx. 15 %, consuming about 10 % of the FPGA's available LUT resources and not impacting the core's critical path at all.

Table 1: FPGA Utilization.

Module	LUTs	FF	BRAM	DSP
`CPU`	3451	1301	4	0
`CI`	530	316	0	3
Total	**4912/5280 (93%)**	**1809/5280 (34%)**	**20/30 (66%)**	**3/8 (37%)**

Table 2: Processing Time Profiling.

	SW-only	SW+CI
`arithmetic mean`	499	77
`std. deviation`	5636	158
frames/s	**0.91**	**13.1**

Future Work

In a next step, further computational bottlenecks that were identified during implementation will be addressed by extending the current CIs. For example, the addition of a third source register operand (i.e. `rs3`) to the CPU's register file only requires two additional FPGA block RAMs and allows implementing R4-type instructions providing more data in parallel to the CI logic. Optimized memory access schemes allowing unaligned data accesses and interrupt driven double-buffering while sampling image data from the off-chip source are further options for increasing overall throughput. Replacing the current base CPU by more performant architecture, which are optimized for embedded image processing like [4], will also be evaluated. Furthermore, a power consumption-driven analysis of the developed hardware extensions is planned.

Summary

This work presents a custom ISA extension of a base RISC-V processor core in order to accelerate the processing of image data primarily focusing on computing arithmetic mean and standard deviation. Three custom instructions were developed motivated by the computational bottleneck of the reference application and implemented for a FPGA prototype. Using the proposed ISA extensions increases the overall frame rate by a factor of 14.4 while expanding the base core's hardware requirements by just 15 %.

References

[1] RISC-V International, "*RISC-V Instruction Set Manual*", github.com/riscv/riscv-isa-manual

[2] S. Nolting et al. "*The NEORV32 RISC-V Processor*", github.com/stnolting/neorv32, doi:10.5281/zenodo.7715920

[3] Lattice Semiconductor, "*iCE40 UltraPlus ML/AI Low-Power FPGAs*", latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus

[4] Fraunhofer IMS, "*The AIRISC RISC-V Processor for embedded AI*", github.com/Fraunhofer-IMS/airisc_core_complex