Automating RISC-V Custom Instruction Integration leveraging High-Level Synthesis

Florian Egert and Bernhard Fischer *

Electronics Design & Integrated Circuits, Siemens

Abstract

Adapting and optimizing systems for maximum performance under tight area and power constraints in a cost-effective way requires design flows that are flexible, provide quick results, and offer automation support to reduce time-intensive and error-prone manual coding. High-Level Synthesis (HLS) is a design process that transforms high-level software algorithms into hardware descriptions on RTL, enabling design exploration and optimization of stringent design constraints. This work presents an automated methodology leveraging HLS for hardware acceleration in RISC-V based open-source cores supporting CV-X-IF, an interface enabling extension of RISC-V cores by custom instructions. This approach, combining hardware synthesis with a processor-agnostic extension interface, enables designers to efficiently accelerate and optimize their diverse applications.

Introduction

Optimizing a system's performance, power, and area (PPA) is a significant challenge in modern computing. The extension of processors with application-specific hardware acceleration is one key strategy to meet specialized workload requirements. Alongside the need for maximized resource efficiency, rapid time-to-market is essential to ensure competitiveness. However, a major limitation to shortening development cycles is the time-consuming and error-prone nature of manual coding. In novel processor architectures like RISC-V, there is a lack of flexible tools for extending processors with custom instructions (CIs) and tailored hardware in an automated fashion.

To address this problem, we propose a RISC-V *Hardware Acceleration Flow* facilitating automated generation and integration of CIs for a broad range of applications. The flow is built using the *High-Level Synthesis (HLS)* tool Catapult HLS [1]. HLS is a technology that synthesizes an RTL implementation in a hardware description language, typically VHDL or Verilog, from an algorithm implemented in a high-level language, such as C, C++, or SystemC (as illustrated in Figure 1). HLS is used in this flow to automatically create an optimized RTL *Wrapper Coprocessor* implementation from a set of one or more software functions from the software application. Migrating these functions from software to hardware in this manner enables system designers to meet performance goals otherwise unachievable with software alone.

The generated Wrapper Coprocessors support the *Core-V eXtension interface (CV-X-IF)* specification of the OpenHW Group [2], making them processor-agnostic as long as the targeted processors support CV-X-IF.

*Corresponding author: `florian.egert@siemens.com`

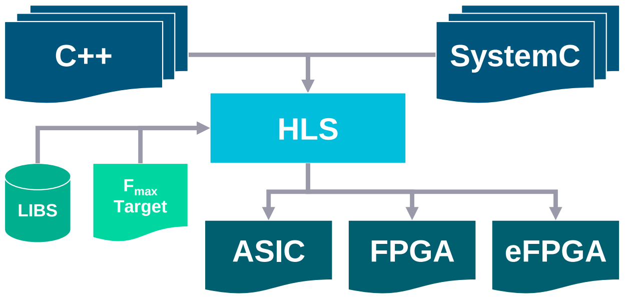

Figure 1: *Concept diagram of High-Level Synthesis (HLS) [1]. The high-level input is represented by the inputs above, design constraints and library inputs on the right, and RTL output below the HLS tool.*

Methodology

The proposed flow is based on our prior work [3] and illustrated in Figure 2. In summary, an algorithm in high-level C/C++ description is partitioned into two parts: One targeting hardware and one targeting software. The hardware part is automatically extended by components for CV-X-IF, and RTL code is automatically created using Catapult HLS. The software algorithm is automatically adapted to call the generated CI hardware and compiled for the RISC-V core.

CI Selection, HLS Adaption The user-selected application in form of a C/C++ algorithm must first be analyzed by the designer to define *accelerator functions*, code snippets suitable for acceleration. The designer marks these code snippets with C/C++ #pragma-statements. After selection, the designer may adapt C/C++ accelerator functions for High-Level Synthesis. Advanced HLS tools are able to synthesize almost all modern C++ constructs that do not dynamically allocate memory. However, certain

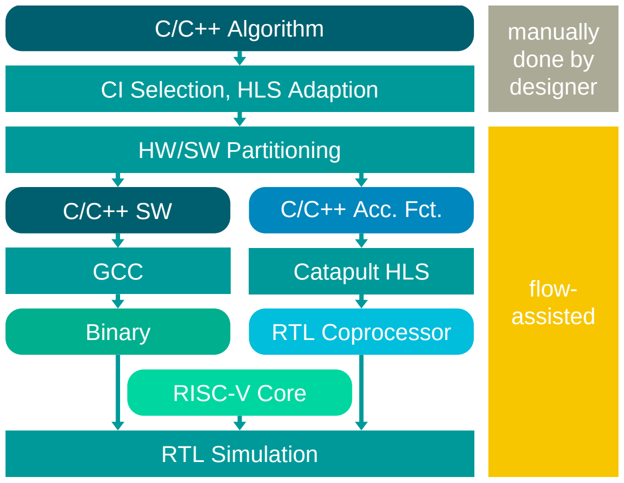

Figure 2: *Flow overview: The squares (left) depict the flow's sub-steps, the round boxes represent their in- and outputs. The boxes to the right indicate steps completed by the designer and flow-assisted steps, respectively.*

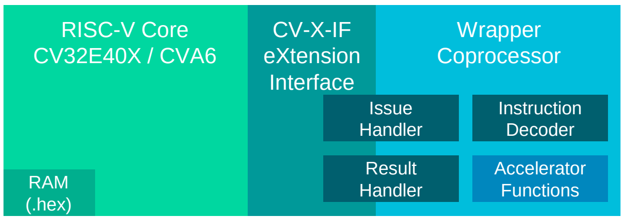

Figure 3: *RTL overview of the wrapper integrated into a resulting system. Core and wrapper are connected via the CV-X-IF, the inner boxes indicate important sub-blocks.*

coding styles, pragmas, and HLS tool settings may be used to achieve RTL implementations comparable to handcrafted designs. In addition, bit accurate data types are used in the high-level language description (C/C++ in case of this flow) so the performed computations will exactly, bit-for-bit, match the synthesized RTL. This also allows the algorithm to be partially validated in the high-level language representation, and for that representation to be used to create expected results for verification of the RTL.

HW/SW Partitioning After these manual steps, the designer uses the flow's HW/SW partitioning tool to split the pragma-annotated C/C++ algorithm into extracted C/C++ accelerator functions and remaining C/C++ software including custom assembly instructions to call these accelerator functions.

SW Compilation, HW Generation The C/C++ software is compiled utilizing GCC. The C/C++ accelerator functions are embedded in a CV-X-IF-compliant C/C++ wrapper description. Based on this description, Catapult HLS generates the RTL wrapper coprocessor and delivers estimations about PPA for specified target systems. The designer may also utilize all capabilities of Catapult, including configuring and optimizing the RTL design and its constraints.

System integration, RTL simulation To verify functionality and timing, the generated RTL wrapper is embedded into a test environment consisting of the wrapper, CV32E40X RISC-V core, and supporting modules. Questa is utilized for RTL simulation of the environment to deliver information about functional correctness and timing details of the application.

Results and Future Work

A major goal of developing the flow is automating manual coding tasks. In our previous work [3] covering CI integration, yet without HLS-based accelerator function generation, we proposed approaches to analyze the potential of working hours reduced when utilizing the flow. The addition of HLS adds an important automation step to this analysis.

The optimization potential of the generated system depends on the input algorithm and selected CIs. While a comprehensive analysis comparing the accelerated systems and software-only solutions running on the CV32E40X is ongoing, first tests with a moderately complex AES-128 use case yield a cycle count reduction of around 65% and a utilization overhead of minimal impact compared to an overall design (e.g., an area score of around 1000 compared to about 30000 for the CVA6 RISC-V core).

In future work, we plan to focus on automating the selection of CIs to leverage more efficient architecture exploration and to identify optimized CI candidates based on the application's requirements.

References

[1] *Catapult C++/SystemC Synthesis*. Siemens. Accessed: March, 2025. [Online]. URL: `https://eda.sw.siemens.com/en-US/ic/catapult-high-level-synthesis/hls/c-cplus/`.

[2] *CORE-V eXtension Interface*. OpenHW Group, 2021. URL: `https://docs.openhwgroup.org/projects/openhw-group-core-v-xif`.

[3] F. Egert et al. "A Methodology for Automating the Integration of User-Defined Instructions into RISC-V Systems based on the CV-X-IF Interface". In: *RISC-V Summit Europe 2024*. 2024. URL: `https://riscv-europe.org/summit/2024/media/proceedings/posters/17_poster.pdf`.

Acknowledgments

The TRISTAN project, nr. 101095947 is supported by Chips Joint Undertaking (CHIPS-JU) and its members and including top-up funding by the Austrian Research Promotion Agency (FFG) and the program "ICT of the Future" of the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK).