

Runtime Reconfiguration of Decoders in Minimal-area RISC-V Cores

Lukas Glantschnig and Tobias Scheipel*

Institute of Technical Informatics, Graz University of Technology

Abstract

Processor implementations designed to occupy minimal areas, such as SERV or FazyRV, are becoming increasingly popular. Some of these designs focus on flexibility and configurability while maintaining their compact design. However, due to their minimal area, implementations often involve compromises in specific components to achieve this level of efficiency. The FazyRV decoder, e.g., is highly optimized for area and therefore omits certain checks for illegal instructions. To address these drawbacks, we propose a concept that uses partial runtime reconfiguration to dynamically replace the decoder’s logic with a more robust variant to enable stricter instruction checking. These modifications introduce an area overhead of up to 39% more flip-flops than the original implementation. Dynamic partial reconfiguration can be triggered during runtime via a memory-mapped register, enabling the processor to continue normal operation seamlessly.

Introduction

In our current, fast-changing world, the Internet of Things (IoT) plays an increasingly important role as more devices in our environment become interconnected and smart [1]. At the same time, flexibility and the ability to reconfigure hardware are becoming necessary to keep up with the pace of change. With billions of devices expected to be deployed in the coming years, efficiency in terms of silicon area, cost, and energy consumption becomes a decisive factor. Scaling down processors is a vehicle to enable large-scale adoption of IoT technology under these constraints. At the end of this scaling, minimal-area processors come to play. Here, the small core footprint is partly achieved through a highly compact decoder. A drawback of implementations like this is that illegal or faulty instructions may cause undefined behavior.

Dynamic Partial Reconfiguration (DPR) offers a promising solution to this dilemma: instead of permanently committing to a single trade-off between efficiency and reliability, the hardware can dynamically adapt its configuration. This allows a processor to operate in a minimal configuration during normal conditions, while switching to a safer configuration when stricter requirements are to be enforced. Such adaptability conveniently combines the strengths of both worlds.

Related Work

There is a steadily increasing number of RISC-V implementations that target minimal area consumption. The most prominent bit-serial processor *SERV* [2] has

demonstrated the smallest RISC-V implementation to date. FazyRV [3] is another approach that features a flexible, scalable datapath architecture. The datapath width can be configured to 1, 2, 4, or 8 bits, which directly influences both the required hardware area and the achievable performance. Other designs, such as *PicoRV32* [4], also achieve very small footprints while maintaining relatively high performance. In parallel, research such as *FlexBex* [5], *moreMCU* [6], or Greyhound [7] has shown how (partial) reconfiguration can be leveraged to add custom instructions at runtime. These concepts demonstrate the use of partial reconfiguration to add custom instructions as hardware extensions dynamically. FlexBex and Greyhound couple an embedded Field-programmable Gate Array (eFPGA) to the core within an Application-specific Integrated Circuit (ASIC), while *moreMCU* follows a soft core FPGA approach. All these concepts enable new functionality to be introduced post-deployment, which is particularly relevant in an era where backward compatibility has become increasingly important.

Implementation

Building upon FazyRV [3], one distinctive feature is its automatically generated instruction decoder. The decoder control signals are obtained using the *Espresso* [8] logic minimization tool. The decoder logic is subsequently derived from a table-like description of the instruction set, which is processed to generate minimized Boolean expressions.

To reduce logic complexity, only the instruction bits necessary to distinguish between valid instructions are explicitly considered. Many other bits are marked as “-” (don’t-care) in the decoder description, as illus-

*Corresponding author: tobias.scheipel@tugraz.at

Input	Output	Instr.
trap_entry, instr, icyc2	Control Signals	
0 <...>---01101-- -	01-0-----00100001000-0000010	LUI
0 <...>---11011-- 0	011-----01100010100-00000-0	JAL
0 <...>---11011-- 1	0-11-----00100010100-0000010	JAL

Table 1: Example of the input format used for Espresso [3].

trated in Table 1. This approach results in a highly compact decoder implementation.

However, this optimization introduces semantic ambiguity. Instructions that differ only in bits marked as don’t-care may produce the *same* control signals as a valid instruction. Consequently, certain illegal or malformed instruction encodings are interpreted as the closest matching valid instruction instead of triggering an exception. This behavior introduces a correctness issue and, in certain scenarios, may pose a security risk.

One possible mitigation is to employ a decoder variant with stricter checking of instruction fields, i.e., by reducing the number of don’t-care assignments. However, such stricter validation typically requires additional logic resources and may therefore negatively impact both area consumption and timing characteristics. To address this trade-off, the Dynamic Reconfiguration Hardware Manager (DRHM) module of *moreMCU* [6] is utilized to enable runtime reconfiguration on FPGAs. The DRHM module allows partial bitstreams to be loaded from memory during system operation – without halting the remainder of the system. This capability enables dynamic switching between different decoder implementations *at runtime*.

Depending on the current application requirements, a larger yet more secure decoder can be loaded as soon as stricter instruction validation is required. In comparison, the smaller optimized decoder can be used when efficiency is the primary concern.

In the presented implementation, reconfiguration can be initiated by the CPU or triggered externally (e.g., via interrupts or a push button for debugging reasons).

Evaluation

To evaluate the resource requirements and performance characteristics of both decoder variants, the existing test and verification infrastructure of FazyRV is reused. When comparing the Lookup Table (LUT) and flip-flop utilization of the modified decoder with those of the original version, a noticeable increase in resource usage is observed. On a Lattice iCE40 FPGA (without DPR capabilities, for comparison reasons), the LUT count increases by approximately 31%, while the number of flip-flops rises by about 39%. For an AMD Artix-7

with DPR capabilities, similar results are obtained. Here, the LUT usage increases by 23%, and the flip-flop count increases by 32%. Although these values are slightly lower than those observed with the iCE40, they still represent a considerable increase in hardware resource consumption.

Based on the timing results, no definitive conclusion can be drawn on whether the decoder itself significantly affects the critical path. In the current system, other parts of the design dominate the timing bottleneck that limits the achievable f_{max} . Yet, systems with complex decoder logic can also profit in this regard.

Conclusion

This work presents a workflow that enables dynamic runtime switch of the FazyRV core’s instruction decoder between the basic, minimal version and an implementation that provides enhanced instruction checking. The approach leverages FPGA DPR, allowing the decoder to be replaced during normal system operation. The evaluation showed that the approach can save resource consumption whenever certain features are not needed, yielding a more efficient system. When using enhanced instruction checking, the hardware overhead is within expected bounds.

References

- [1] Anita Choudhary. “Internet of Things: a comprehensive overview, architectures, applications, simulation tools, challenges and future directions”. In: *Discover Internet of Things* 4.31 (Dec. 2024). ISSN: 2730-7239.
- [2] Olof Kindgren. *SERV: The world’s smallest RISC-V CPU*. <https://github.com/olofk/serv>. Accessed: 2025-09-06. 2023.
- [3] Meinhard Kissich and Marcel Baunach. “FazyRV: Closing the Gap between 32-Bit and Bit-Serial RISC-V Cores with a Scalable Implementation”. In: *Proc. of the 21st ACM Int’l Conf. on Computing Frontiers*. Association for Computing Machinery, May 2024, pp. 240–248.
- [4] Claire Wolf. *PicoRV32 – A Size-Optimized RISC-V CPU*. <https://github.com/YosysHQ/picorv32>. Accessed: 2023-10-30. 2023.
- [5] Nguyen Dao et al. “FlexBex: A RISC-V with a Reconfigurable Instruction Extension”. In: *2020 Int’l Conf. on Field-Programmable Technology (ICFPT)*. 2020.
- [6] Tobias Scheipel, Florian Angermair, and Marcel Baunach. “moreMCU: A Runtime-reconfigurable RISC-V Platform for Sustainable Embedded Systems”. In: *Proc. of the 25th Euromicro Conf. on Digital System Design (DSD)*. Maspalomas, Spain, 2022, pp. 24–31.
- [7] Leo Moser et al. “Greyhound: A Reconfigurable and Extensible RISC-V SoC and eFPGA on IHP SG13G2”. In: *Proc. of the 33rd Austrochip Workshop on Microelectronics (Austrochip)*. Linz, Austria, 2025. ISBN: 2689-8144.
- [8] Robert K. Brayton et al. *Logic Minimization Algorithms for VLSI Synthesis*. Boston, MA: Kluwer Academic Publishers, 1984.