Challenges for Embedded Machine Learning on Custom RISC-V ASIPs

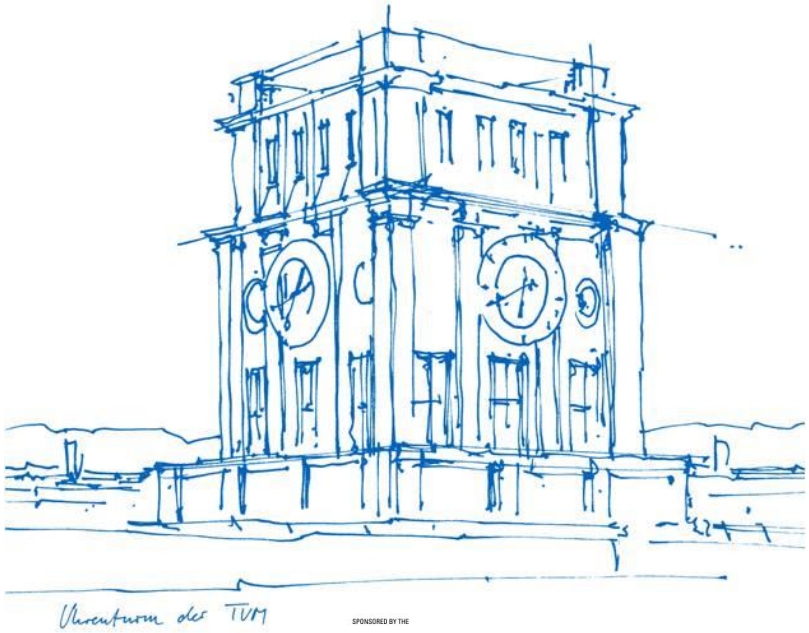
Philipp van Kempen

Technical University of Munich

TUM School of Computation, Information and Technology

Chair of Electronic Design Automation

Munich, 23th April 2024



This work has been developed in the project MANNHEIM-FlexKI. MANNHEIM-FlexKI is funded by the German Ministry of Education and Research (BMBF) (reference numbers: 01IS22086A-L). The authors are responsible for the content of this publication.

SPONSORED BY THE
Federal Ministry of Education and Research

Content

1. Motivation / Scenario
2. Overview of Challenges
3. Solutions
4. Conclusion / Outlook

Embedded Machine Learning

- Machine learning workloads are emerging rapidly → Resource demands are rising exponentially

- Conventional (cloud-based) approach has lots of disadvantages
  - Power consumption (CO_2 footprint)
  - Latency (not suitable for real-time applications)
  - Privacy

  → Run Inferences on-device

- Focus: Extreme Edge Devices (TinyML)
  - Many methodologies could also be applied to more powerful HW later

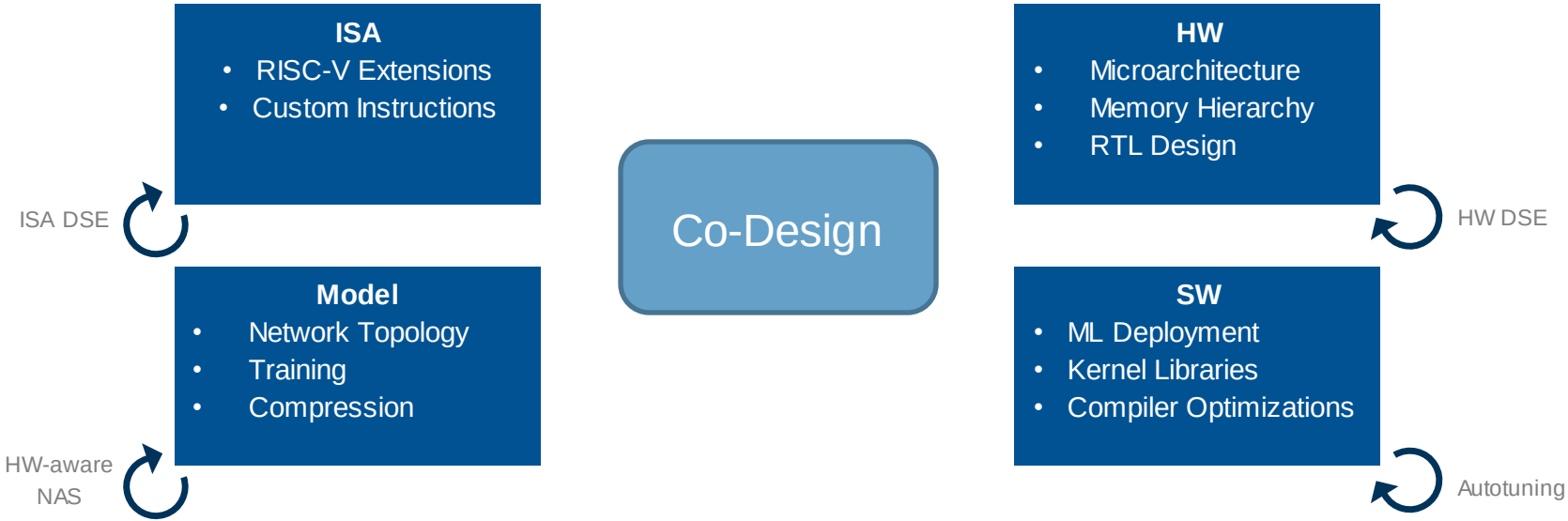
ASIP Design

- There exist numerous off-the-shelf platforms for embedded ML
  - Edge TPU (Google)
  - Jetson Nano (NVIDIA)

- For low-power applications (MCUs) these are not applicable
  - General purpose MCUs are lacking support for efficient TinyML operations
  - Specialize hardware (ISA+Microarchitecture) to targeted application
  - → Design Application Specific Instruction-Set Processors (ASIPs)

- Approaches:
  - Manual
  - HW/SW-Codesign driven (High-level Synthesis)
  - Mixed

Our Vision


ISA
RISC-V Extensions
Custom Instructions
ISA DSE
Model
Network Topology
Training
Compression
HW-aware NAS
Co-Design
HW
Microarchitecture
Memory Hierarchy
RTL Design
HW DSE
SW
ML Deployment
Kernel Libraries
Compiler Optimizations
Autotuning

Overview of Challenges

Challenge	Description	Solution
Availability	*Missing Hardware, Specifications,…*	
Benchmarking		
Profiling		
Validation		
Tuning		
Retargeting		

Open Source!

Challenge: Availability

Problem: Hardware for testing on-device is not available (or yet to be designed)

- Example: RISC-V Vector Extension (RVV v1.0)
  - Ratified in Late 2021, ramp-up for HW using RVV takes at least 2-3 years
  - Numerous HW released with old specification → Avoid if possible
  - First commercial development board showed up in early 2024

Solution: Virtual Prototyping

- [ETISS]: Extendable Translating Instruction Set Simulator
- [CoreDSL]: Describe processor cores at the level of their instruction set architecture
- [CorePerfDSL]: Modelling of pipeline/microarchitecture for performance estimation

Overview of Challenges

Challenge	Description	Solution
Availability	*Missing Hardware, Specifications,…*	*Virtual Prototyping (ETISS & CoreDSL)*
Benchmarking	*Comparison between different Frameworks, Targets,… difficult*	
Profiling		
Validation		
Tuning		
Retargeting		

Open Source!

Challenge: Benchmarking

Problem: Running comparable benchmarks on X targets, Y frameworks, Z toolchains, U Models

- Need a single tool for each target/framework/toolchain/model instead of hardcoded scripts

Solution: End-to-End TinyML Deployment and Benchmarking Flow

- [MLIF] (Machine Learning Interface)
  - Framework/target-independent abstraction layers for Target SW
- [MLonMCU]
  - Provides support for
    - 15+ targets (mainly RISC-V simulators)
    - 6 backends ([TVM] and TFLM)
  - Handling of Dependencies
  - Analysis and Exploration methods
  - Designed with parallelism/reproducibility in mind

Overview of Challenges

Challenge	Description	Solution
Availability	*Missing Hardware, Specifications,…*	*Virtual Prototyping (ETISS & CoreDSL)*
Benchmarking	*Comparison between different Frameworks, Targets,… difficult*	*TinyML Deployment & Benchmarking Flow (MLonMCU)*
Profiling	*Where to apply optimizations? How to find bottlenecks?*	
Validation		
Tuning		
Retargeting		

Work in progress!

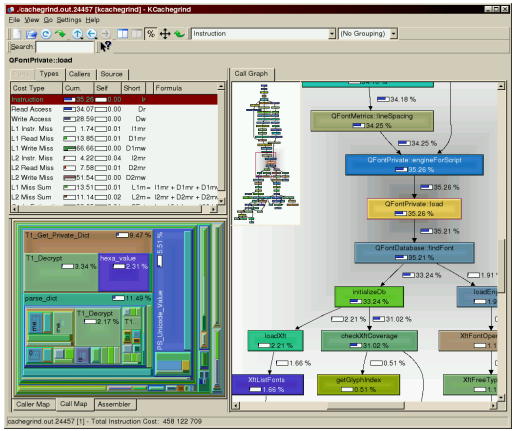
Challenge: Profiling

Problem: Do not waste time optimizing irrelevant parts of a program

- Example:
  - Optimized NN-kernel to be 2x faster → End-to-End inference speed only improves by 1%?
  - 95% of the total runtime is spent in different layers

Solution: Multi-level profiling/tracing methodology and bottleneck detection

- Convert RTL/ISS traces into Gprof/Callgrind compatible format
  → Allows to use existing tools for analysis
- Multiple Events: Instructions, Cycles, Cache Misses, Branches,…
- Annotation of sources at various abstraction layers
  → ML Layers, C/C++, LLVM-IR, Assembly



Overview of Challenges

Challenge	Description	Solution
Availability	*Missing Hardware, Specifications,…*	*Virtual Prototyping (ETISS & CoreDSL)*
Benchmarking	*Comparison between different Frameworks, Targets,… difficult*	*TinyML Deployment & Benchmarking Flow (MLonMCU)*
Profiling	*Where to apply optimizations? How to find bottlenecks?*	*Multi-level profiling based on static/dynamic binary/trace analysis*
Validation	*How to guarantee Model accuracy is maintained during deployment?*	
Tuning		
Retargeting		

Open Source!

Challenge: Validation

Problem: We have to maintain original model's accuracy during deployment

- Degradations in accuracy can happen at several stages
  - Compression (Quantization/Pruning)
  - ML Compiler / Kernels → Rounding, Overflows,…
  - Bugs (SW Compiler, ISA, RTL)

Solution: Dataset-based automatic validation framework integrated in MLonMCU

- Target-independent (MLIFIO abstraction layer)
- Highly configurable (Supported Metrics, Thresholds,…)
- Automated and efficient (minimal runtime/resource overheads)

Overview of Challenges

Challenge	Description	Solution
Availability	*Missing Hardware, Specifications,…*	*Virtual Prototyping (ETISS & CoreDSL)*
Benchmarking	*Comparison between different Frameworks, Targets,… difficult*	*TinyML Deployment & Benchmarking Flow (MLonMCU)*
Profiling	*Where to apply optimizations? How to find bottlenecks?*	*Multi-level profiling based on static/dynamic binary/trace analysis*
Validation	*How to guarantee Model accuracy is maintained during deployment?*	*Target-independent Validation Flow in MLonMCU (MLIFIO)*
Tuning	*Can we optimize the generated Kernels for specific targets efficiently?*	
Retargeting		

Work in progress!

Challenge: Tuning

Problem: How to get target-optimized ML kernels

- Default (fallback) TVM kernels should be avoided
- Writing hand-optimized kernels (see CMSIS-NN) is infeasible

Solution: Perform on-device autotuning with TVM

- Available tuners: AutoTVM, AutoScheduler, MetaSchedule
- Improvements required to improve the tuning process
  - Reliability
  - Cost Models (target-aware & workload-aware)
  - Efficiency

Overview of Challenges

Challenge	Description	Solution
Availability	*Missing Hardware, Specifications,…*	*Virtual Prototyping (ETISS & CoreDSL)*
Benchmarking	*Comparison between different Frameworks, Targets,… difficult*	*TinyML Deployment & Benchmarking Flow (MLonMCU)*
Profiling	*Where to apply optimizations? How to find bottlenecks?*	*Multi-level profiling based on static/dynamic binary/trace analysis*
Validation	*How to guarantee Model accuracy is maintained during deployment?*	*Target-independent Validation Flow in MLonMCU (MLIFIO)*
Tuning	*Can we optimize the generated Kernels for specific targets efficiently?*	*TVM's MetaScheduler with improvements*
Retargeting	*How to avoid manual efforts to support new target hardware?*	

Retargeting

Definition

“In software engineering, retargeting is an attribute of software development tools that have been specifically designed to generate code for more than one computing platform.”

→ **Here:** Retargetable Compilers

Types of Compilers

- SW Compilers (LLVM, GCC) → See next slides
- ML Compilers (TVM) → Planned
- HW Compilers

Introducing Seal5

Seal5 - Semi-automated LLVM Support for RISC-V ISA Extensions (Including Autovectorization)

Inputs

- CoreDSL code for custom instructions
- Optional: YAML Settings

Outputs

- Patched LLVM Toolchain

```
CV_SDOTSP_H {
        encoding: 5'b10101 :: 1'b0 :: 1'b0 :: rs2[4:0] :: rs1[4:0] :: 3'b000 :: rd[4:0] :: 7'b1111011;
        assembly: "{name(rd)}, {name(rs1)}, {name(rs2)}";
        behavior: {
                if (rd != 0) X[rd] = (unsigned<32>)((signed)(
                        X[rd] +
                        ((signed)X[rs1][15: 0] * (signed)X[rs2][15: 0]) +
                        ((signed)X[rs1][31:16] * (signed)X[rs2][31:16])));
        }
}
```

Retargeting Support Levels (LLVM)

Tool	**Assembler** (Encoding, Format, Effects,…)	**Intrinsics/Builtins** (LLVM-IR, C/C++)	**CodeGen** (ISel Patterns, Legalization)	**Auto-Vectorization** (SIMD, Heuristics,…)
Extensible Compiler [DLR]	✅ (Needs user inputs)	✅	❌	❌
- [TUDA]	✅	✅	❌	❌
OpenASIP 2.0 [TUNI]	✅	✅	❌	❌
Ours [Seal5]	✅	✅ (Experimental)	✅ (Semi-automated)	✅ (Narrow 32-bit SIMD only)

Usage by SW Developer:

```
asm("mac x3, x4, x5");
```

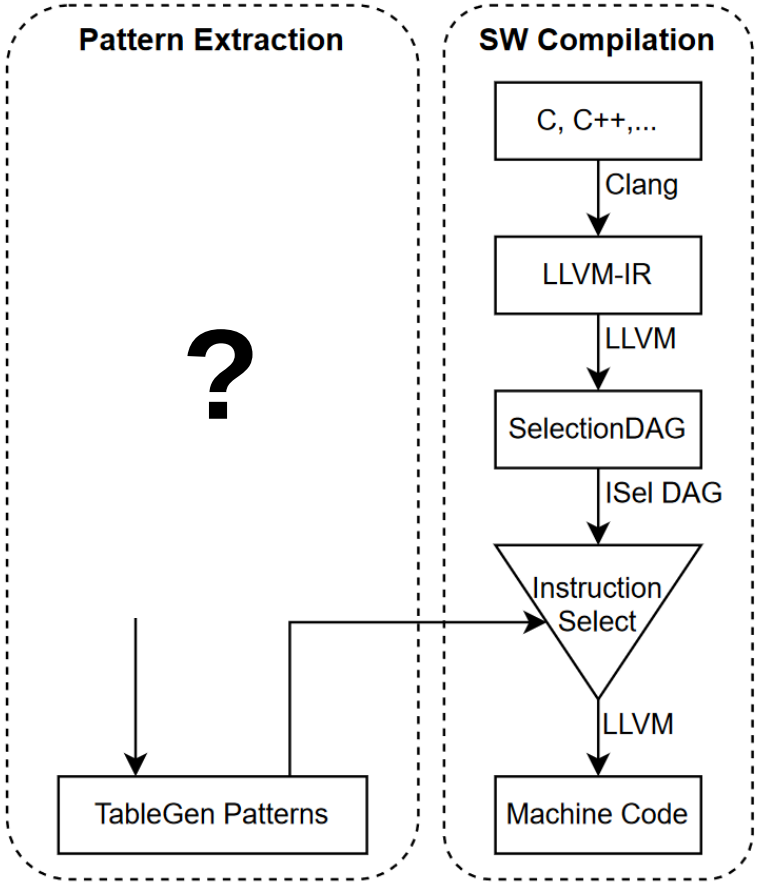
```
__builtin_mac(acc, x, y);
```

```
acc += x * y;
```

```
for (i = 0; i < n; i++) {
  acc += arr_x[i] * arr_y[i];
}
```

Why do we need patterns?

- During compilation the original program is lowered to intermediate representations (IRs) in a step-by-step fashion
- Optimizations are applied along the way
- During *Instruction Selection* Generic LLVM instructions are converted to target-specific *MachineInstructions*
- Instruction Selection depends on manually specified patterns to insert any instructions.



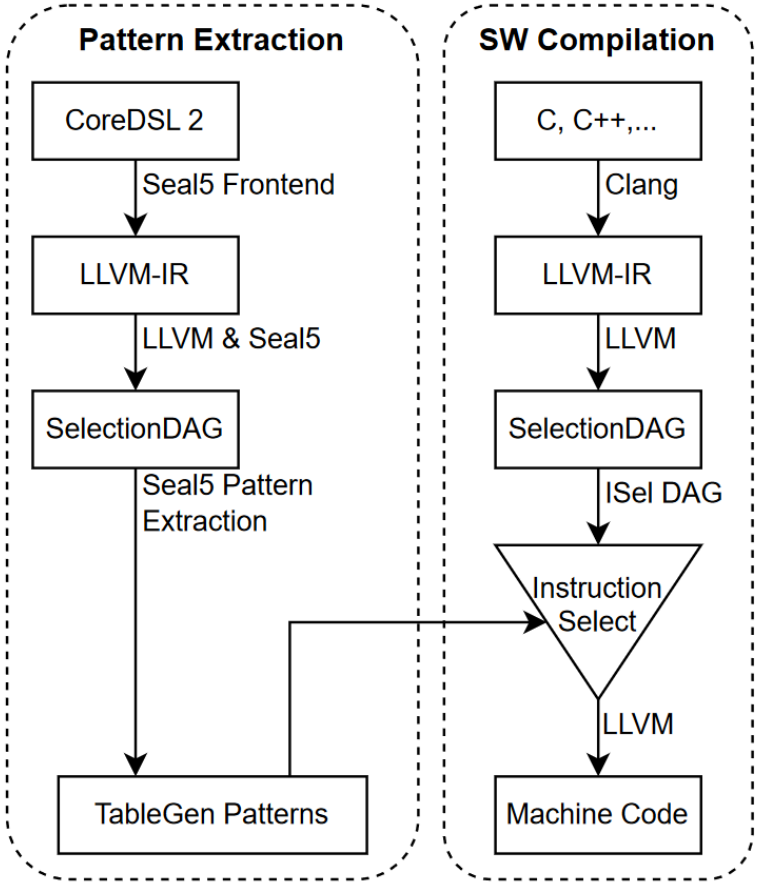
Generating ISel Patterns

Method

1. Convert CoreDSL behavior to LLVM-IR functions
2. Perform lowering in a similar way to target SW
3. Add hook to emit final DAG right before Instruction Selection would take place
4. Transform DAG nodes to TableGen code for patterns

Advantages

- Re-use existing code in LLVM
- Same optimizations → increased likeliness that extracted patterns will actually match
- SIMD-instructions are detected automatically

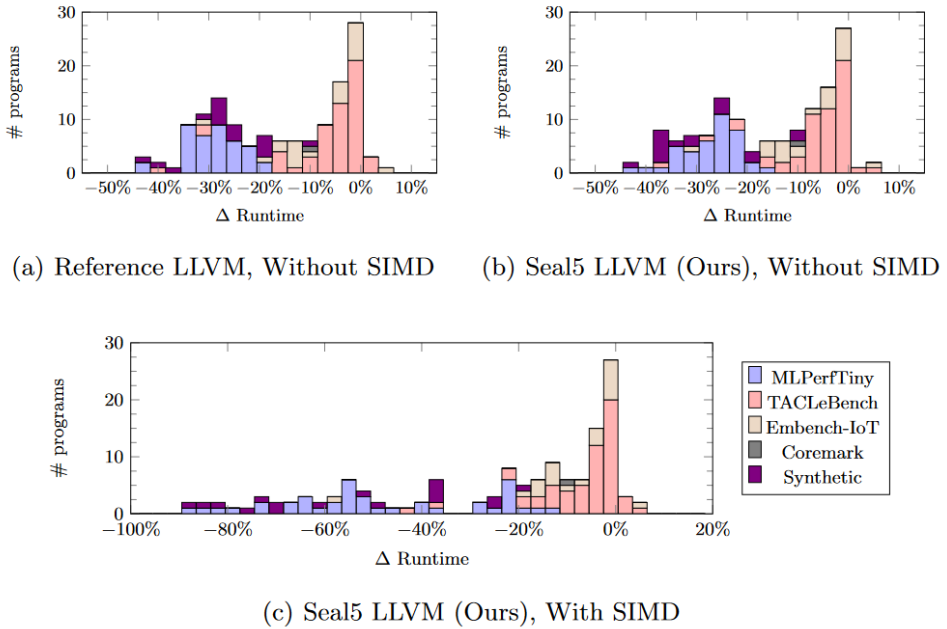


Seal5 Evaluation (Core-V)

Core-V Extension (OpenHW Group)
- 300+ ALU/Mem/SIMD/… instructions
- Implemented in [CV32E40P]

Configurations
1. Baseline (RV32IM)
2. Core-V Reference
3. Seal5 Generated
   a) Without SIMD
   b) With SIMD

Benchmarks
- 100+ embedded programs



(a) Reference LLVM, Without SIMD (b) Seal5 LLVM (Ours), Without SIMD

(c) Seal5 LLVM (Ours), With SIMD

Fig. 3. Histogram of the reduction of all benchmark program's runtime in cycles measured on the CV32E40P core. Baseline is the runtime of the program compiled without any Core-V extension support. Δ Runtime smaller 0% indicates that the LLVM with Core-V instruction extension could improve runtime compared to a program with just standard RISC-V instructions. Programs are grouped by benchmark (colors) into runtime bins.

Overview of Challenges

Challenge	Description	Solution	
Availability	*Missing Hardware, Specifications,…*	*Virtual Prototyping (ETISS & CoreDSL)*	Analysis
Benchmarking	*Comparison between different Frameworks, Targets,… difficult*	*TinyML Deployment & Benchmarking Flow (MLonMCU)*	Analysis
Profiling	*Where to apply optimizations? How to find bottlenecks?*	*Multi-level profiling based on static/dynamic binary/trace analysis*	Analysis
Validation	*How to guarantee Model accuracy is maintained during deployment?*	*Target-independent Validation Flow in MLonMCU (MLIFIO)*	Analysis
Tuning	*Can we optimize the generated Kernels for specific targets efficiently?*	*TVM's MetaScheduler with improvements*	Automation
Retargeting	*How to avoid manual efforts to support new target hardware?*	*Metamodel-based Generation of LLVM and TVM Patches*	Automation

Further challenges

Not covered in this talk:

- General Constraints of Embedded Systems

Needs to be considered at all design stages

- ML Specifics
  - Model Design & Model Compression

Starting with pre-trained and quantized Models

- HW Specifics
  - RTL-generation for ASIPs (HW/SW-Codesign)

Required to estimate implementation overhead of optimizations

- Real-world problems
  - Flash time bottleneck & wearing out hardware

Conclusion

Summary

- Without overcoming aforementioned challenges the HW/SW co-exploration (DSE) will be infeasible
- Retargeting is essential to eliminate manual efforts

Seal5 – Retargeting LLVM Compiler for RISC-V

- Novel approach for robust pattern generation and SIMD support
- Compared with reference Core-V vendor toolchain

Next steps:

- Retargeting support for ML Compilation
- Solve remaining challenges

References

[TUDA] Thesis: Halkenhäuser, M. *Automatic Compiler Support for Application-Specific Instruction Set Architecture Extensions* (Master's thesis, Technische Universität).

[DLR] Paper: Schlamelcher, J., & Grüttner, K. (2022). A DSL based approach for supporting custom RISC-V instruction extensions in LLVM.
Repo: https://github.com/DLR-SE/extensible-compiler

[TUNI] Paper: Hepola, K., Multanen, J., & Jääskeläinen, P. (2022, July). OpenASIP 2.0: co-design toolset for RISC-V application-specific instruction-set processors. In *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)* (pp. 161-165). IEEE.
Repo: https://github.com/cpc/openasip

[CV32E40P] Website: OpenHW Group CV32E40P User Manual - https://cv32e40p.readthedocs.io/en/latest

[ETISS] Paper: Mueller-Gritschneder, Daniel, et al. "The extendable translating instruction set simulator (ETISS) interlinked with an MDA framework for fast RISC prototyping." *Proceedings of the 28th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*. 2017.
Repo: https://github.com/tum-ei-eda/etiss

[CoreDSL] Paper: Emrich, Karsten, et al. "A Flexible Simulation Environment for RISC-V." *RISC-V Summit Europe*. 2023.
Repo: https://github.com/Minres/CoreDSL

[CorePerfDSL] Paper: Foik, Conrad, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. "CorePerfDSL: A Flexible Processor Description Language for Software Performance Simulation." *2022 Forum on Specification & Design Languages (FDL)*. IEEE, 2022.
Repo: N/A

[MLIF] Repo: https://github.com/tum-ei-eda/mlonmcu-sw

[MLonMCU] Paper: van Kempen, Philipp, et al. "MLonMCU: TinyML Benchmarking with Fast Retargeting." *arXiv preprint arXiv:2306.08951* (2023).
Repo: https://github.com/tum-ei-eda/mlonmcu

[TVM] Paper: Chen, Tianqi, et al. "TVM: end-to-end optimization stack for deep learning." *arXiv preprint arXiv:1802.04799* 11.2018 (2018): 20.
Repo: https://github.com/apache/tvm

[Seal5] Paper: N/A
Repo: Release in May 2024