Longnail High-Level Synthesis of Portable Custom Instruction Set Extensions for RISC-V Processors from Descriptions in the Open-Source CoreDSL Language

TECHNISCHE UNIVERSITÄT DARMSTADT

Julian Oppermann[1*], Brindusa Mihaela Damian-Kosterhon[1], Florian Meisel[1], Tammo Mürmann[1], Eyck Jentzsch[2], Andreas Koch[1]

MINRES TECHNOLOGIES

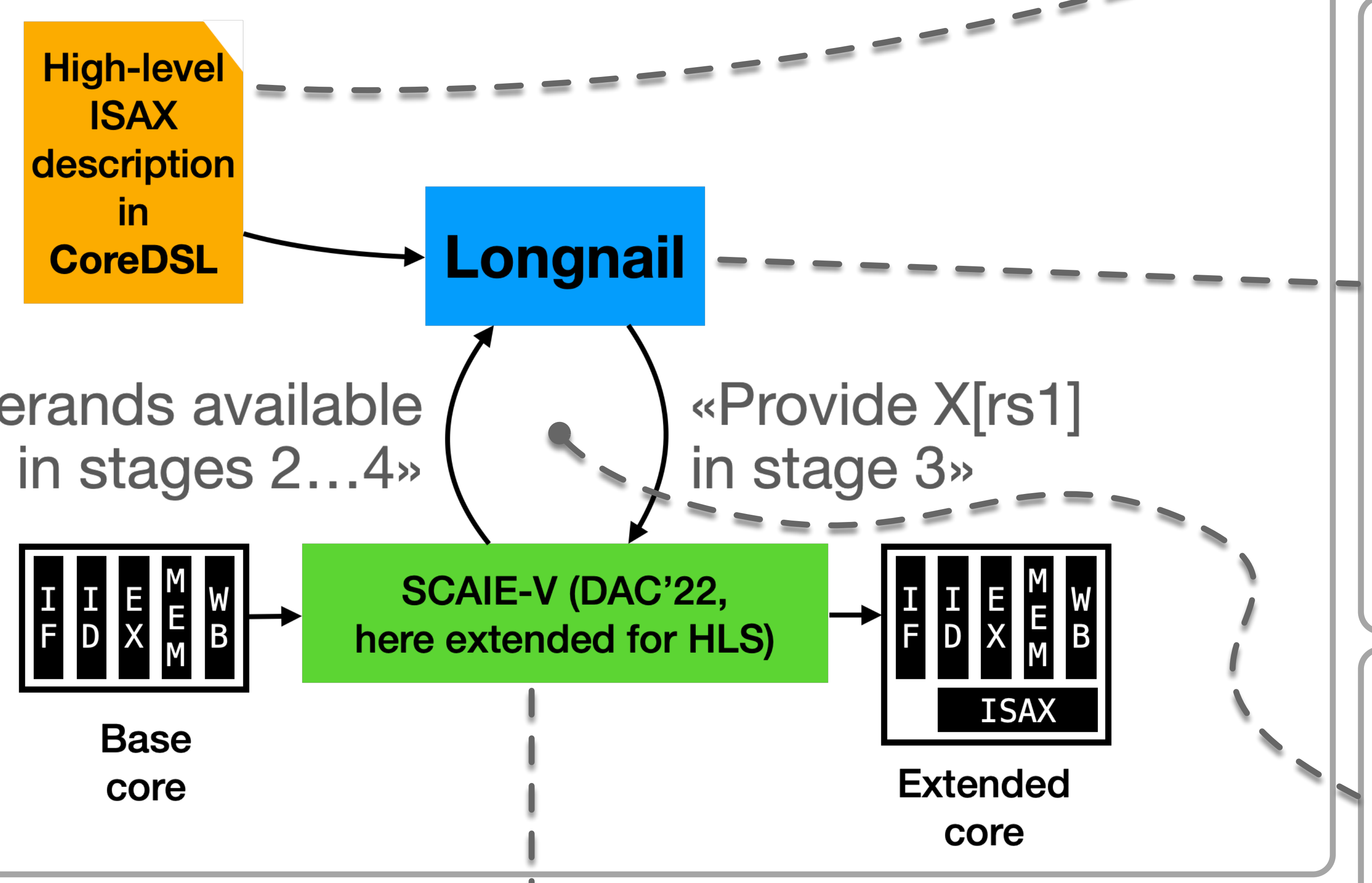
[1] Technical University of Darmstadt, [2] MINRES Technologies GmbH, * now at Codeplay Software

Introduction

- Modern embedded, IoT devices are expected to run ML, signal processing, etc.
- ISA extensions ("**ISAX**") → cost-effective + energy-efficient way to accelerate applications on generic base cores
- RISC-V ecosystem: 💯 for ISAX approach
- But in practice: only limited reuse and exploration
  - Implementing an extension manually: hard
  - Existing solutions are vendor-specific, not portable across cores or microarchitectures

Idea: Accessible and portable ISAX design

- *Longnail:* Microarchitecture-agnostic high-level synthesis
  - *CoreDSL:* new user-friendly language
- Bi-directional communication with *SCAIE-V* interface generator
- Automatic integration into base core


High-level ISAX description in CoreDSL
Longnail
«Operands available in stages 2…4»
«Provide X[rs1] in stage 3»
IF ID EX MEM WB
SCAIE-V (DAC'22, here extended for HLS)
IF ID EX MEM WB ISAX
Base core
Extended core

CoreDSL — new ISAX design language

- Intuitive ADL with C-inspired syntax & concise structure
- Bitwidth-aware type system to prevent implicit loss of precision
- Control-flow constructs & ISAX-specific syntax extensions

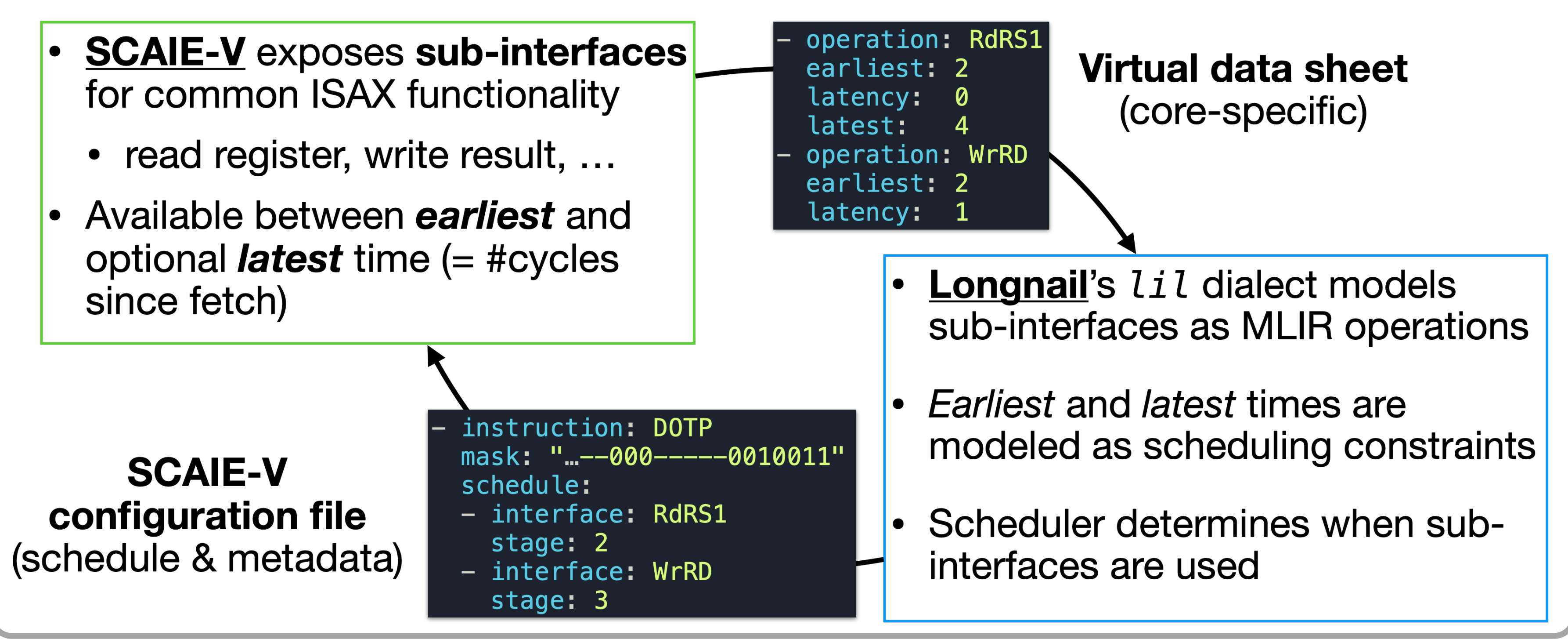
```
import "RV32I.core_desc"

InstructionSet X_DOTP extends RV32I {
  instructions {
    DOTP {
      encoding: 7'd0 :: rs2[4:0] :: rs1[4:0] ::
                3'd0 ::  rd[4:0] :: 7'b0001011;
      behavior: {
        signed<32> res = 0;
        for (int i = 0; i < 32; i += 8) {
          signed<16> prod = (signed) X[rs1][i+7:i] *
                            (signed) X[rs2][i+7:i];
          res += prod;
        }
        X[rd] = (unsigned) res;
} } } }
```

Longnail — HLS for ISAXes

- Built from scratch on top of MLIR and CIRCT
- Gradual lowering using upstream and custom dialects
  - `coredsl`: instructions, always-blocks, registers, etc.
  - `lil`: control-dataflow graphs of combinational logic
- Custom scheduling problem to target SCAIE-V-supported core + ILP-based scheduler
  - Respects availability of interfaces
  - Minimizes latency and lifetimes of intermediate values
- Construct ISAX module in CIRCT's dialects, emitted in SystemVerilog

Bi-directional communication



- **SCAIE-V** exposes **sub-interfaces** for common ISAX functionality
  - read register, write result, …
- Available between ***earliest*** and optional ***latest*** time (= #cycles since fetch)

Virtual data sheet (core-specific)

```
- operation: RdRS1
  earliest: 2
  latency:  0
  latest:   4
- operation: WrRD
  earliest: 2
  latency:  1
```

- **Longnail**'s `lil` dialect models sub-interfaces as MLIR operations
- *Earliest* and *latest* times are modeled as scheduling constraints
- Scheduler determines when sub-interfaces are used

SCAIE-V configuration file (schedule & metadata)

```
- instruction: DOTP
  mask: "…--000-----0010011"
  schedule:
  - interface: RdRS1
    stage: 2
  - interface: WrRD
    stage: 3
```

ISAXes beyond R-type instructions

- Two novel language constructs in CoreDSL:
  - *always*-block: Execute behavior continuously, independently of fetched instructions
  - *spawn*-block: Other instructions can be executed in parallel to spawned behavior
- Extended SCAIE-V tool provides interface to the core
  - handles data hazards and arbitration, provides access to program counter, instantiates custom registers

```
InstructionSet zol extends RV32I {
  architectural_state {
    register unsigned<32> START_PC, END_PC, COUNT;
  }
  always {
    zol {
      // program counter (`PC`) defined in RV32I
      if (COUNT != 0 && END_PC == PC) {
        PC = START_PC;
        --COUNT;
} } } }
```

```
InstructionSet sqrt extends RV32I {
  instructions {
    sqrt {
      encoding: 12'd0 :: rs[4:0] :: 3'b000 :: rd[4:0]
                :: 7'b0101011;
      behavior: {
        unsigned<32> arg = X[rs], res = 0;
        spawn {
          for (int i = 0; i < 32; ++i) {/* CORDIC */}
          X[rd] = res;
} } } } }
```

Impact in Scale4Edge ecosystem

- CoreDSL successfully used by application engineers to accelerate audio event detection application
- Successful tapeout with earlier version of SCAIE-V and handwritten ISAX module → 15 % area for ISAX enables real-time performance

see: Ecker et al., A Scalable RISC-V Hardware Platform for Intelligent Sensor Processing, DATE 2024

Results

- 8 ISAXes, 4 base cores
- Demonstrated end-to-end flow from CoreDSL description to RTL
- ISAXes can be composed

	ORCA		Piccolo		PicoRV32		VexRiscv (5 stage)	
	Area	Freq.	Area	Freq.	Area	Freq.	Area	Freq.
Base core, area excluding any caches	6,612 μm^2	996 MHz	26,098 μm^2	420 MHz	4,745 μm^2	1,278 MHz	9,052 μm^2	701 MHz
autoinc	+ 20 %	- 6 %	+ 3 %	- 9 %	+ 23 %	+ 0 %	+ 12 %	+ 2 %
dotprod	+ 23 %	- 14 %	+ 4 %	+ 0 %	+ 21 %	- 2 %	+ 21 %	+ 2 %
ijmp	+ 2 %	- 3 %	+ 7 %	+ 3 %	+ 7 %	+ 2 %	+ 12 %	+ 0 %
sbox	+ 7 %	- 2 %	+ 0 %	+ 3 %	+ 6 %	+ 2 %	+ 8 %	- 1 %
sparkle	+ 85 %	- 24 %	+ 2 %	- 1 %	+ 46 %	+ 0 %	+ 45 %	- 2 %
sqrt_tightly	+ 80 %	- 32 %	+ 22 %	- 15 %	+ 100 %	- 5 %	+ 43 %	- 8 %
sqrt_decoupled	+ 56 %	- 5 %	+ 10 %	+ 3 %	+ 111 %	- 7 %	+ 47 %	+ 6 %
… without data-hazard handling	+ 46 %	- 6 %	+ 10 %	+ 3 %	+ 96 %	- 2 %	+ 40 %	+ 4 %
zol	+ 7 %	- 2 %	+ 13 %	+ 4 %	+ 10 %	- 1 %	+ 14 %	- 3 %
autinc+zol	+ 29 %	- 6 %	+ 3 %	+ 2 %	+ 32 %	- 1 %	+ 16 %	+ 5 %

Paper

CoreDSL frontend & specification

SCAIE-V 2.0

Contact: {oppermann, damian, meisel, muermann, koch}@esa.tu-darmstadt.de, eyck@minres.com

Supported by the German Federal Ministry of Education and Research in the projects "Scale4Edge" (grants: 16ME0122K-140, 16ME0465, 16ME0900, 16ME0901) and "MANNHEIM-FlexKI" (grant: 01IS22086A-L).

Scale4Edge

MANNHEIM FlexKI