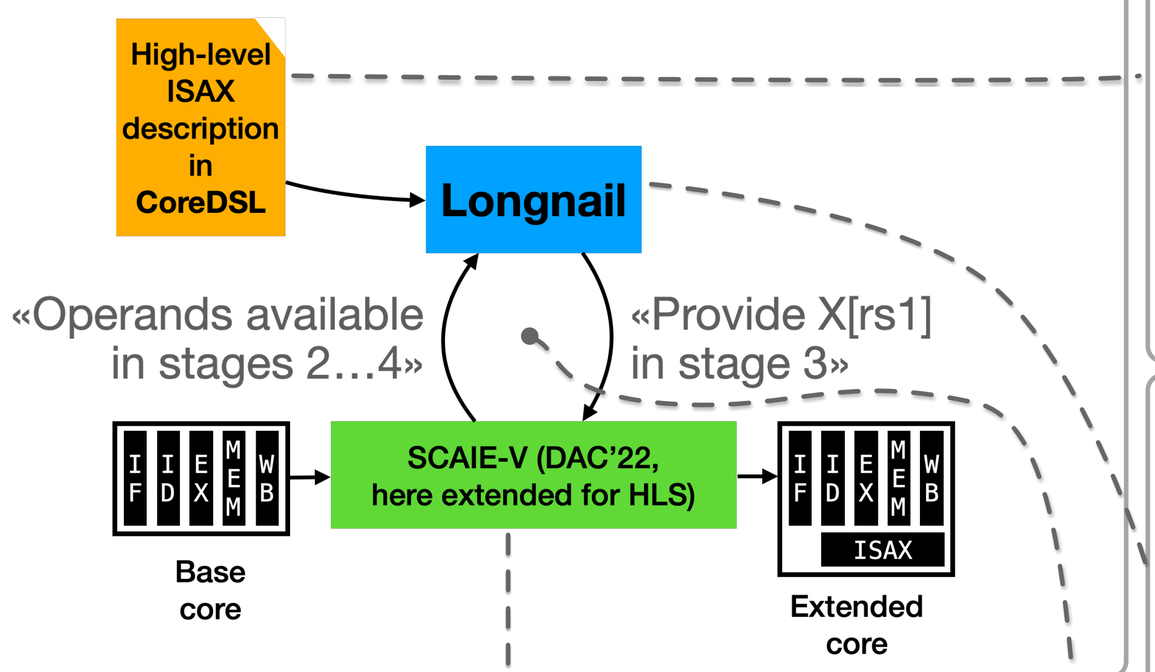
Hardware Synthesis of CoreDSL Custom Instructions for MCU- and Application-Class Cores

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

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

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



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

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;
} } } } }
```

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)

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

Virtual data sheet (core-specific)

- **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
```

NEW: Resource Sharing

- Modulo scheduling based resource sharing:
  - Sharing between different instructions
  - Within a single instruction
- Vector ISAX: with dot product and element-wise multiply

22nm ASIC flow Area excluding SRAMs	VexRiscv (5 Stage) Area	Freq.
Base core	9,052 µm²	701 MHz
No Sharing	+ 204 %	- 8 %
II = 1	+ 215 %	- 4 %
II = 2	+ 170 %	- 8 %
II = 3	+ 122 %	- 1 %

Resource Allocation	#Add	#Mul
No Sharing	2	6
II = 1	2	3
II = 2	1	2
II = 3	1	1

Instruction Latency	Vec-Dot	Vec-Mul
No Sharing	3 cycles	1 cycles
II = 1	3 cycles	1 cycles
II = 2	3 cycles	2 cycles
II = 3	4 cycles	3 cycles

NEW: Application-class cores: CVA5, CVA6

- Our flow supports two more cores:
  - **CVA5**: 4 stages, register renaming, variable-latency FUs
  - **CVA6**: 6 stages, scoreboarding, variable-latency FUs
- Full ISAX compatibility, single dedicated FU for SCAIE-V
  - *Semi-coupled* mode: Pipelined ISAX → Pipelined FU (+reordering)
  - *Decoupled* mode (CVA5): Instruction can retire before writeback
- Custom registers: Hazard handling adapts to pipeline layout

Results

- 8 ISAXes, 6 base cores
- Demonstrated end-to-end flow from CoreDSL to RTL
- ISAXes can be composed

22nm ASIC flow Area excluding SRAMs	ORCA Area	ORCA Freq.	Piccolo Area	Piccolo Freq.	PicoRV32 Area	PicoRV32 Freq.	VexRiscv (5 stage) Area	VexRiscv (5 stage) Freq.	CVA5 Area	CVA5 Freq.	CV64A6 Area	CV64A6 Freq.
Base core	6,612 µm²	996 MHz	26,098 µm²	420 MHz	4,745 µm²	1,278 MHz	9,052 µm²	701 MHz	58,460 µm²	770 MHz	103,271 µm²	731 MHz
autoinc	+ 20 %	- 6 %	+ 3 %	- 9 %	+ 23 %	+ 0 %	+ 12 %	+ 2 %	+ 5 %	- 1 %	- 2 %	- 1 %
dotprod	+ 23 %	- 14 %	+ 4 %	+ 0 %	+ 21 %	- 2 %	+ 21 %	+ 2 %	+ 3 %	- 0 %	- 2 %	- 9 %
ijmp	+ 2 %	- 3 %	+ 7 %	+ 3 %	+ 7 %	+ 2 %	+ 12 %	+ 0 %	+ 5 %	- 6 %	- 2 %	+ 0 %
sbox	+ 7 %	- 2 %	+ 0 %	+ 3 %	+ 6 %	+ 2 %	+ 8 %	- 1 %	+ 1 %	- 0 %	- 1 %	+ 1 %
sparkle	+ 85 %	- 24 %	+ 2 %	- 1 %	+ 46 %	+ 0 %	+ 45 %	- 2 %	+ 4 %	- 2 %	- 2 %	+ 1 %
sqrt_tightly	+ 80 %	- 32 %	+ 22 %	- 15 %	+ 100 %	- 5 %	+ 43 %	- 8 %	+ 4 %	- 1 %	+ 0 %	- 2 %
sqrt_decoupled	+ 56 %	- 5 %	+ 10 %	+ 3 %	+ 111 %	- 7 %	+ 47 %	+ 6 %	+ 7 %	+ 1 %	-	-
. . . without handling data-hazards	+ 46 %	- 6 %	+ 10 %	+ 3 %	+ 96 %	- 2 %	+ 40 %	+ 4 %	+ 8 %	- 2 %	-	-
zol	+ 7 %	- 2 %	+ 13 %	+ 4 %	+ 10 %	- 1 %	+ 14 %	- 3 %	+ 9 %	- 2 %	- 1 %	+ 3 %
autoinc+zol	+ 29 %	- 6 %	+ 3 %	+ 2 %	+ 32 %	- 1 %	+ 16 %	+ 5 %	+ 7 %	- 3 %	+ 1 %	+ 0 %

Paper **CoreDSL frontend & specification** **SCAIE-V 2.0**

Contact: {oppermann, damian, meisel, muermann, mueller, 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).
This work has received funding from the Chips Joint Undertaking (CHIPS-JU) TRISTAN project under grant agreement No 101095947. CHIPS JU receives support from the European Union's Horizon Europe research and innovation programme.