Generation of scalar ASIPs with OpenVADL

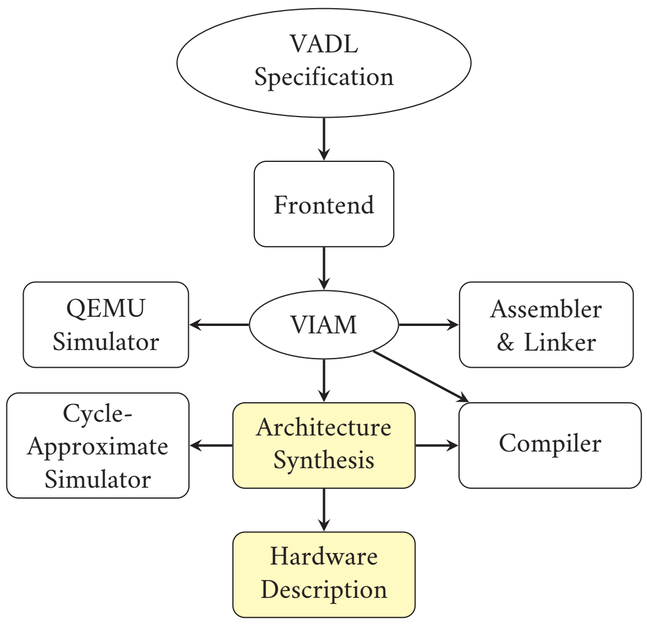
Linus Halder, Matthias Raschhofer, Andreas Krall
Technische Universität Wien, Vienna, Austria
vadl@tuwien.ac.at

OpenVADL

OpenVADL[1] is an open-source implementation of the Vienna Architecture Description Language (VADL), a **processor description language**.

Alongside a **hardware description** (this work), OpenVADL can also generate various tools from a VADL specification:

- assembler and linker
- LLVM compiler
- QEMU instruction set simulator



Overview of the OpenVADL architecture

The generators use a common frontend and a shared intermediate representation, the VADL intermediate architecture model (VIAM). This representation models behavior as graphs.

Hardware Generator

VADL separates the specification of the instruction set architecture (ISA) and the microarchitecture (MiA).

The microarchitecture defines

- Instruction fetch logic
- A sequence of pipeline stages
- Instruction behavior included in these stages

```
import isa::{RV32I}

micro architecture ThreeStage implements RV32I = {

  stage IF -> ( fr : FetchResult ) = {
    fr := fetchNext       // fetch next instruction
  }

  stage ID -> ( ir : Instruction ) = {
    let instr = decode( IF.fr ) in {
      instr.read( @X )    // read from X registers
      ir := instr         // stage output
    }
  }

  stage EX -> ( ir : Instruction ) = {
    let instr = ID.ir in {
      instr.read( @MEM )  // read from memory
      instr.write( @X )   // write to X registers
      ir := instr
    }
  }
}
```

VADL specification of a simple three-stage pipeline

Architecture Synthesis

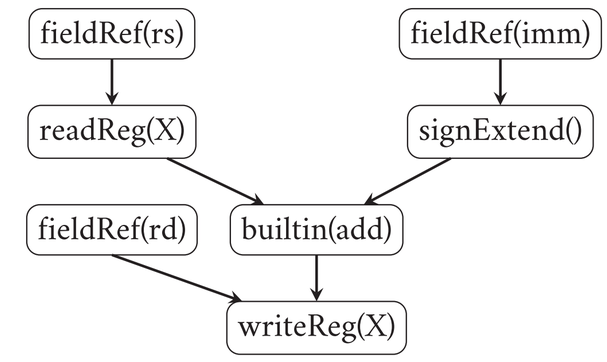
This step first combines the behavior of all instructions into a single graph, annotating all nodes with the set of instructions it is used in. Then the nodes are placed into a stage according to the MiA specification.

To reduce the number of read and write operations in the graph, all nodes with disjoint sets of instructions are merged. For this, select nodes are introduced at their inputs.

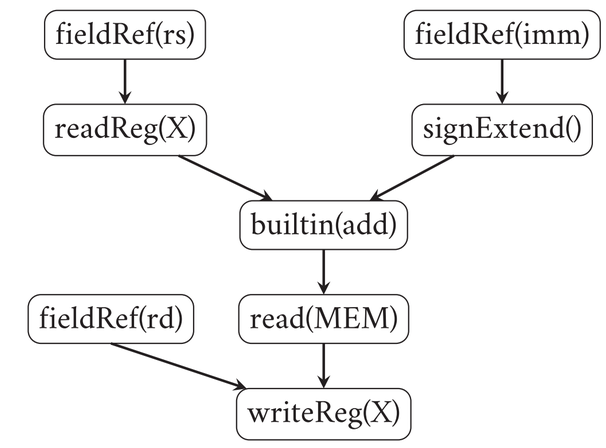
During architecture synthesis potential data hazards are analyzed and forwarding logic is generated. Forwarding logic can be adapted to include specific forwarding paths in the MiA specification.

The architecture synthesis concludes with the generation of control logic that outputs enable signals for the stages and the pipeline registers. These pipeline registers are required for all edges that cross between stages. This yields the **MiA implementation** consisting of:

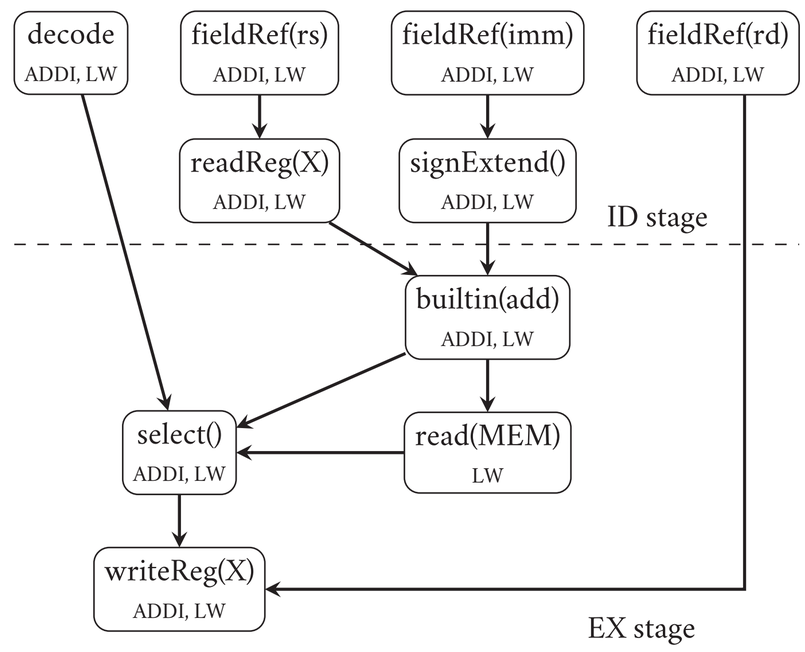
- Implementations for the stages, including pipeline registers
- Control and forwarding logic connected to the stages



ADDI instruction behavior



LW instruction behavior



Pipelined behavior graph

Hardware Description

The MiA implementation now only contains elements that can be emitted in a hardware description language (HDL). The hardware generator uses Chisel as its HDL.

The generator produces source files for

- Each stage
- Control and forwarding modules
- Register files

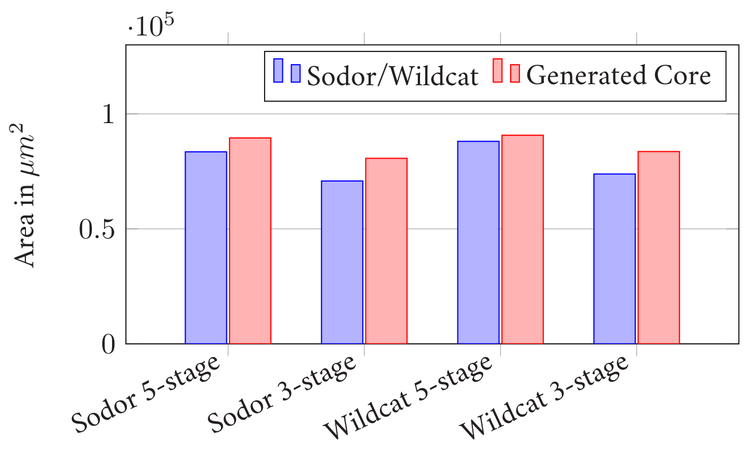
Evaluation

To demonstrate the efficiency of the hardware generator, we compare the area and maximum clock frequency of synthesized RISC-V RV32I cores. We selected two educational cores:

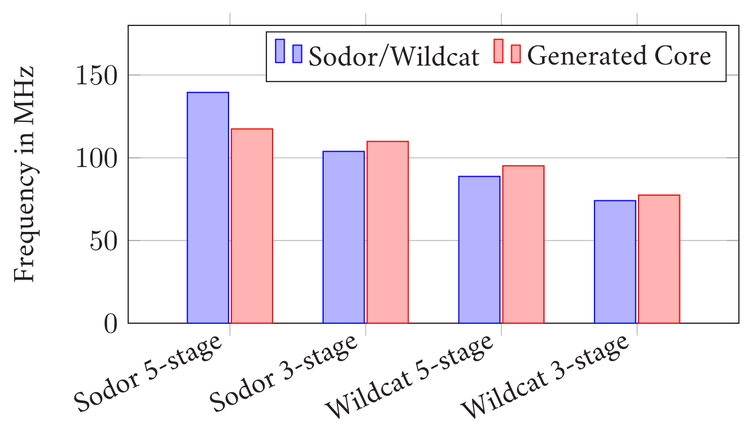
- Sodor
- Wildcat

The control and status registers were removed from both cores, because they have very different ranges of implemented features. Synthesis was done with Yosys and OpenROAD using the SkyWater Foundry SKY130 high density standard cell library.

The OpenVADL hardware generator can achieve similar area and frequency compared to existing designs, while also providing other tools for the ISA in use, such as a compiler and an instruction set simulator.



Placement area of synthesized RV32I cores



Maximum clock frequency of synthesized RV32I cores

[1] https://github.com/openVADL