Open-Source Development Platform for RISC-V Application-Specific Instruction-Set Processors

Kari Hepola Joonas Multanen Pekka Jääskeläinen

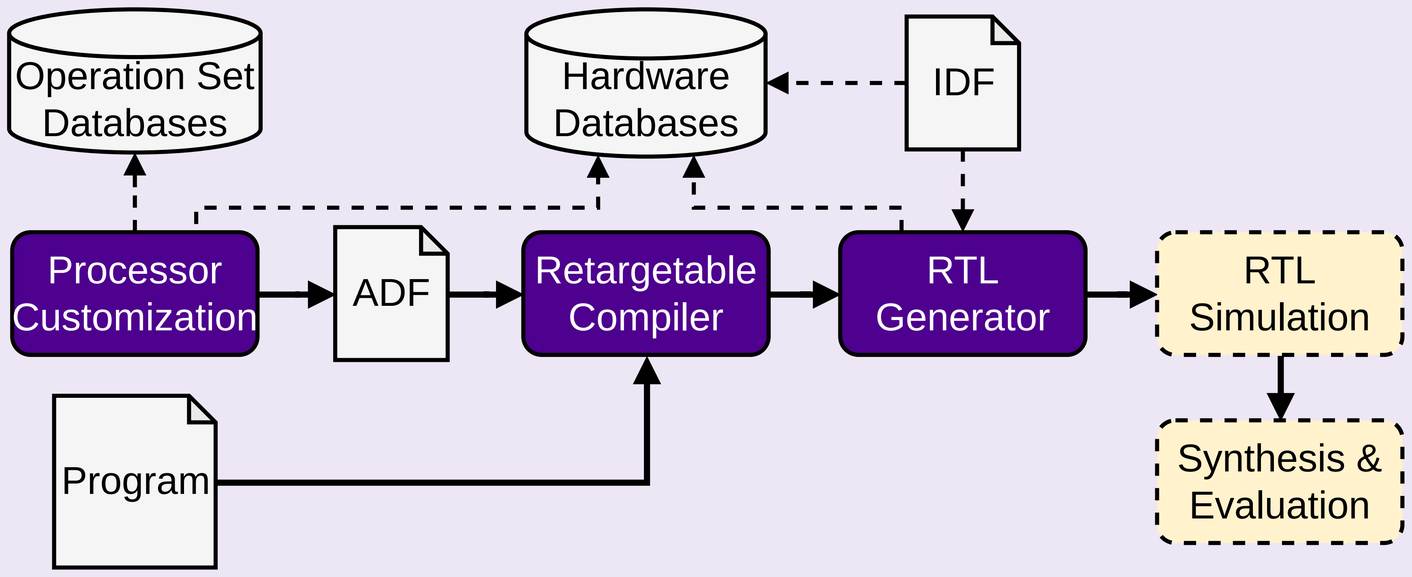
Tampere University, Tampere, Finland

Application-Specific Instruction-Set Processors

- *Application-specific instruction-set processors* (**ASIPs**) offer a middle ground in terms of performance, energy efficiency and flexibility between fixed-function accelerators and general-purpose processors.
- The performance and energy efficiency benefit is acquired by customizing the instruction for a set of target applications, while still maintaining programmability via instruction-set based control.
- **OpenASIP** is an ASIP co-design toolset developed at Tampere University since the early 2000s. It has been used in various research and commercial design cases, and has recently been extended to support the RISC-V instruction set architecture.

Customization Flow in OpenASIP

- The customization flow is built on top of an *architecture definition file* (**ADF**) that is used to describe the instruction set of the processor.
- The operation semantics are described separately in operation set databases to adapt a hierarchical description with a high degree of reuse.
- Both retargetable compilation and hardware generation are supported with separate tools for each step of the design space exploration.



Compiler Retargetability

- The toolset ships with a retargetable **LLVM**-based RISC-V compiler that adapts to the architecture definition.
- Custom instructions can be called via intrinsics in the source code, which are processed in a custom pass in the backend phase of the compilation. The backend pass replaces the custom instruction with an encoding that matches the generated hardware.

```
unsigned sha_sig0(unsigned x) {
 unsigned a = (x << 25) | (x >> 7);
 unsigned b = (x << 14) | (x >> 18);
 unsigned c = (x >> 3);
 return a ^ b ^ c;
}
```

Standard RISC-V Compiler

```
sha_sig0:
  srli    a1, a0, 7
  slli    a2, a0, 25
  or      a1, a1, a2
  srli    a2, a0, 18
  slli    a3, a0, 14
  or      a2, a2, a3
  xor     a1, a1, a2
  srli    a0, a0, 3
  xor     a0, a0, a1
  ret
```

```
unsigned sha_sig0(unsigned x) {
  _OA_RV_SHA_SIG0(x,0,x);
  return x;
}
```

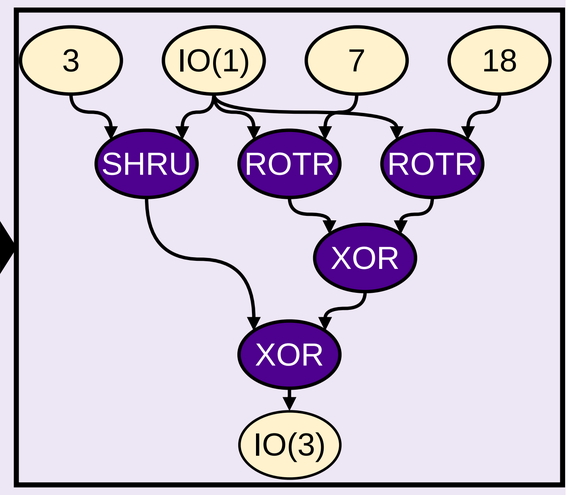
OpenASIP

```
sha_sig0:
  sha_sig0 a0, a0, zero
  ret
```

Operation Descriptions

- *Directed acylic graph* (**DAG**) representation makes it possible to describe operations as a chain of already implemented operations.
- Alternatively, the designer can describe operation schemantics directly with HDL-snippets.

```
Var a, b, c, d;
OP(ROTR, IO(1), 7, a);
OP(ROTR, IO(1), 18, b);
OP(SHRU, IO(1), 3, c);
OP(XOR, a, b, d);
OP(XOR, d, c, IO(3);
```



Processor Generator

- The hardware description is generated based on the ADF, operation set and hardware databases.
- Generation of components, such as decoder and instruction fetch, are separated from the function unit generation, which varies heavily between architectures.
- **FUGen** creates register pipelines based on the operation latency and chains basic operations of DAG-based operation-descriptions and connects HDL operation descriptions to the function unit interface.