



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

**Kari Hepola**, Joonas Multanen and Pekka Jääskeläinen

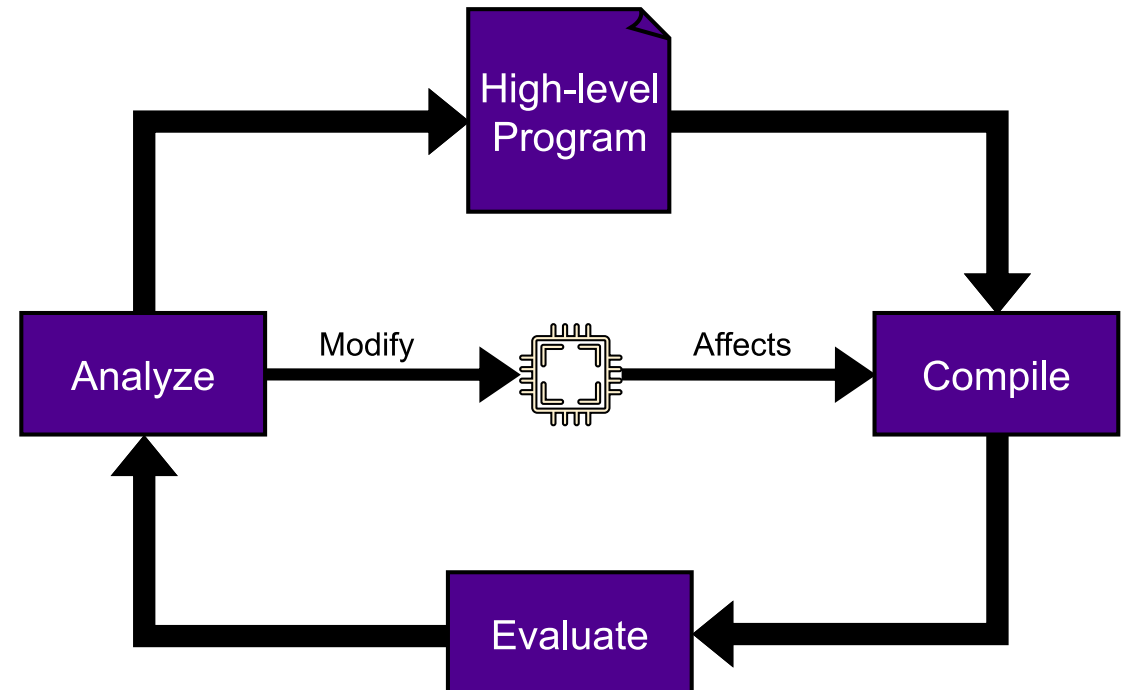
Customized Parallel Computing (CPC) Group, Tampere University, Finland

<https://www.tuni.fi/cpc>

<https://github.com/cpc/openasip>

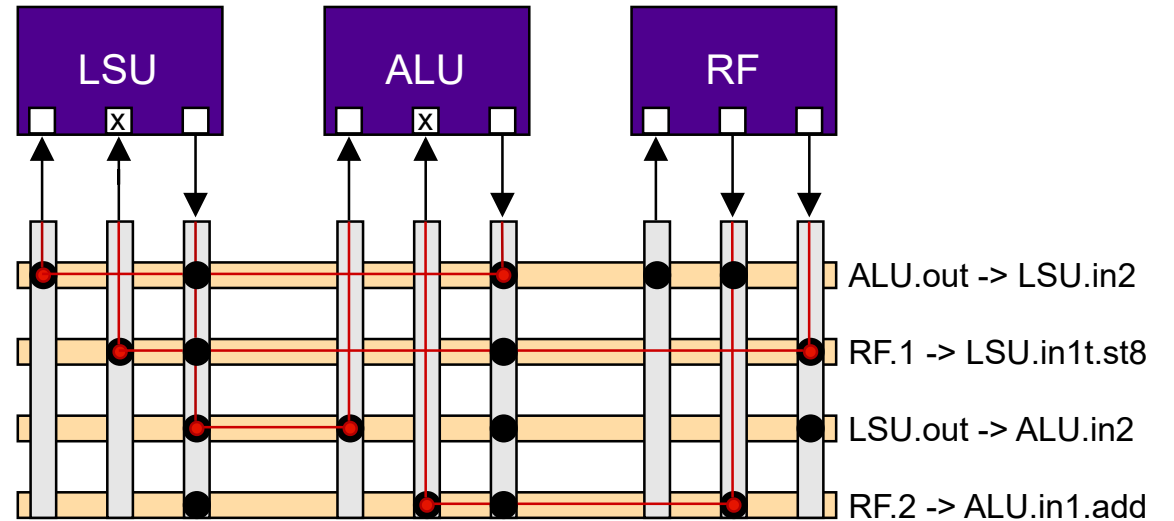
# Background

- Application-specific instruction-set processors (**ASIPs**)
- Co-design toolsets are often proprietary
- Demand for customized processors
  - Better energy efficiency and performance for a set of target applications
  - HW/SW co-design toolsets make customization and programming easier



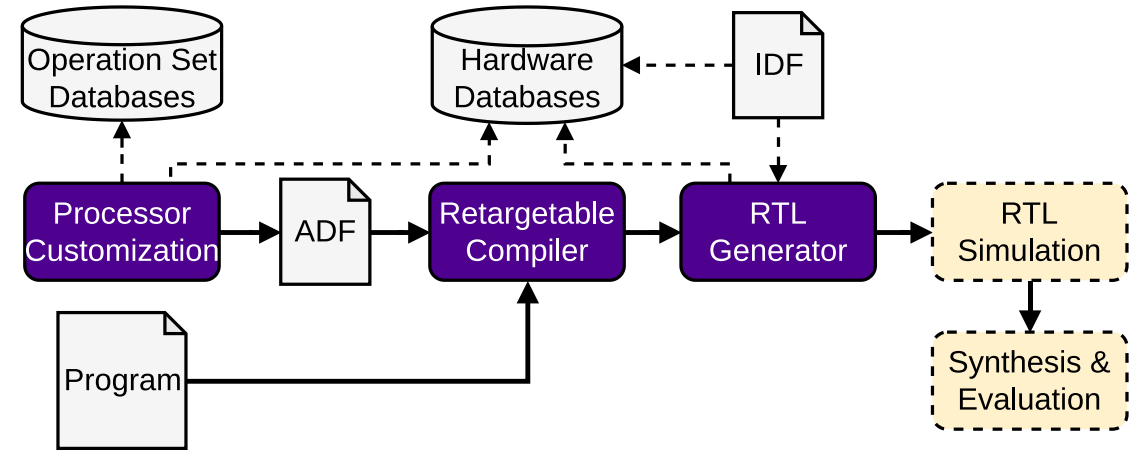
# OpenASIP Toolset

- Architecture template
  - Transport triggered architecture (TTA)
  - Low-level architecture with high scheduling freedom
- Open source
  - Toolset - LGPL v2.1
  - Generated hardware - MIT
- Extensive support for TTA-based ASIPs
  - Custom operations
  - Retargetable LLVM-based compiler
  - Hardware generation



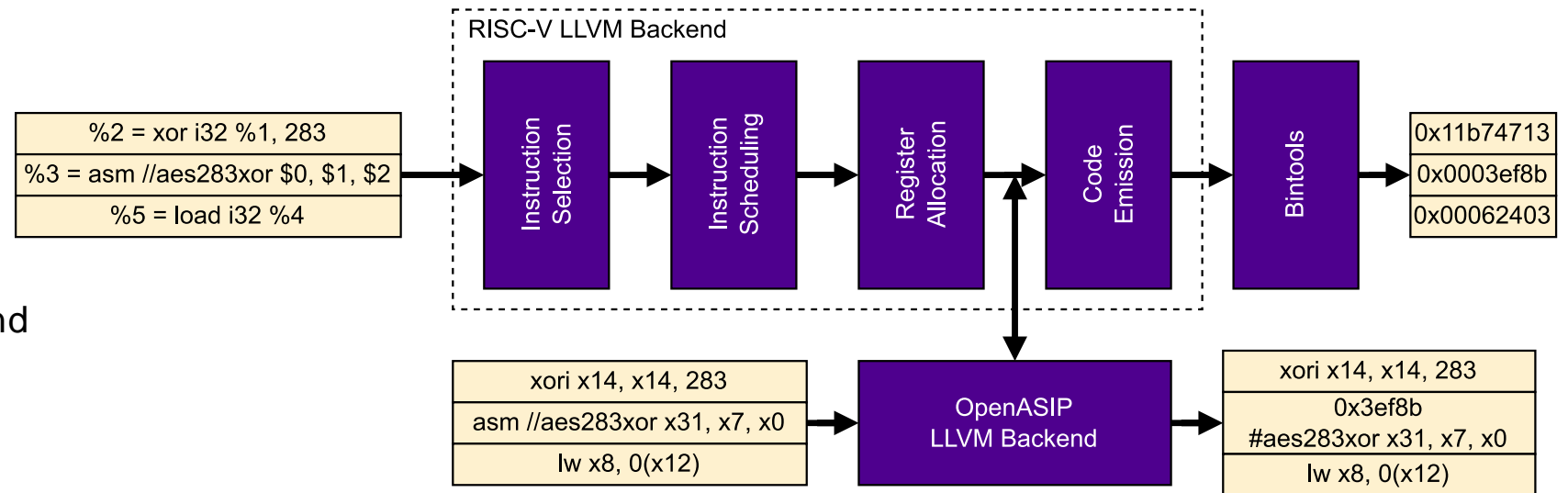
# RISC-V Support in OpenASIP

- Extended for RISC-V ASIPs
- Support for RISC-V custom instructions
  - Instruction intrinsics
  - Retargetable compiler
  - RTL generation
- Rapid hardware generation flow



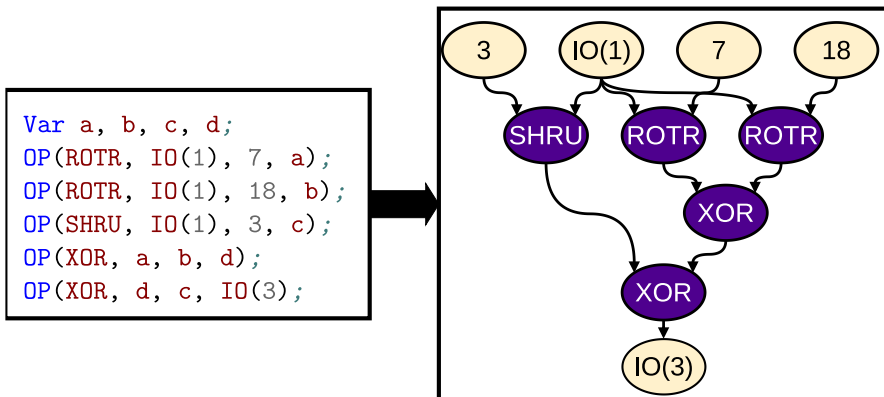
# Compiler Adaptation

- LLVM RISC-V backend
- OpenASIP backend as a dynamic library
- Preemit pass
  - Replace intrinsics with opcode and operand binaries
- Backend generation work in progress
  - Allows automatic instruction selection



# Operation Descriptions

- Operation DAGs (Directed Acyclic Graphs)
  - Describe a custom operation with a chain of already implemented operations



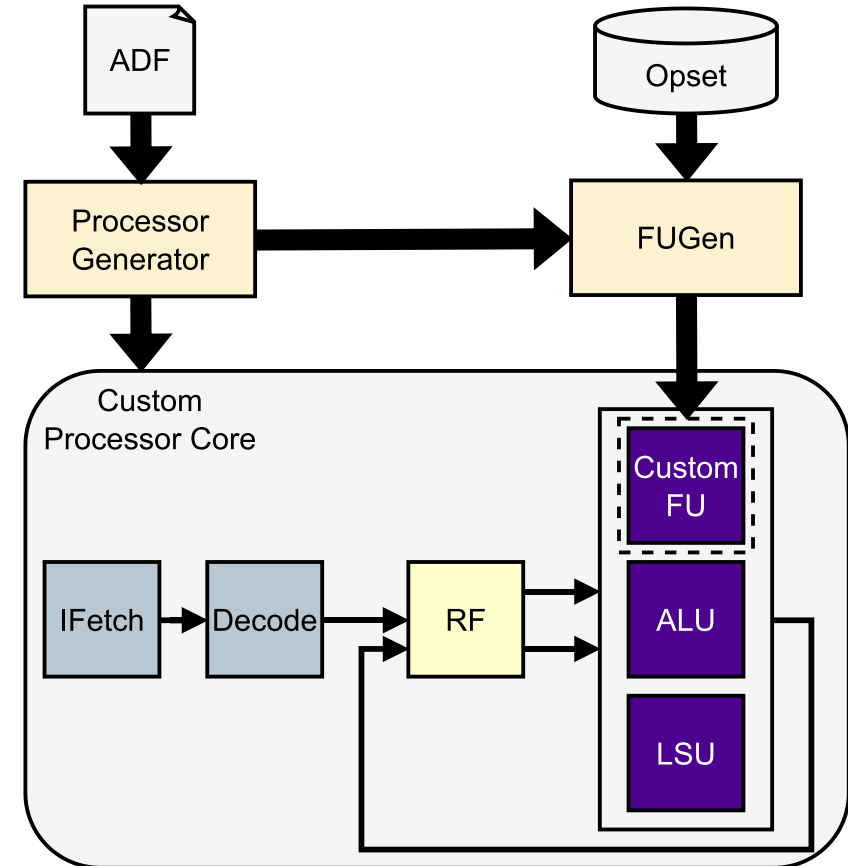
- HDL snippets
  - Add a HDL snippet that describes the semantics of the operation
  - Practical when implementing bit manipulation operations

```

for bits in 0 to 7 loop
    op3(bits) <= op1(7-bits);
end loop;
    
```

# FUGen

- Integrated with OpenASIP processor generation
- Chains operations in DAG-based descriptions
- Connects signals of HDL snippets to the function unit IO
- Automatically adds pipeline registers based on the architecture definition



# Example: SHA256

- Design an ASIP for SHA256
- Design steps:
  1. Investigate the application for custom operation candidates
  2. Add the operation descriptions to an operation set library
  3. Add the new operations to the processor description
  4. Add intrinsic calls to the application source code

1. Identify operation candidates



2. Add new operation entries



3. Modify architecture description



4. Modify application source code



# 1. Identify Custom Operation Candidates

- Often easy for the application developer
- Program hot spots
- Bit manipulations
- Hardware complexity

sha256sig0 :  $rotr^7(in) \oplus rotr^{18}(in) \oplus (in \gg 3)$   
sha256sig1 :  $rotr^{17}(in) \oplus rotr^{19}(in) \oplus (in \gg 10)$   
sha256sum0 :  $rotr^2(in) \oplus rotr^{13}(in) \oplus rotr^{22}(in)$   
sha256sum1 :  $rotr^6(in) \oplus rotr^{11}(in) \oplus rotr^{25}(in)$

# 2. Adding New Operation Descriptions

## Create a new entry

**Operation properties**

Operation properties  
Name: sha256sig0

Reads memory  Writes memory  
 Can trap  Has side effects  
 Clocked

Operation description

Affected by

operation

ABS Add Delete

Operation inputs

operand	type	element width	element
1	SintWord	32	1
2	SintWord	32	1

Add... Modify... Delete

Affects

operation

ABS Add Delete

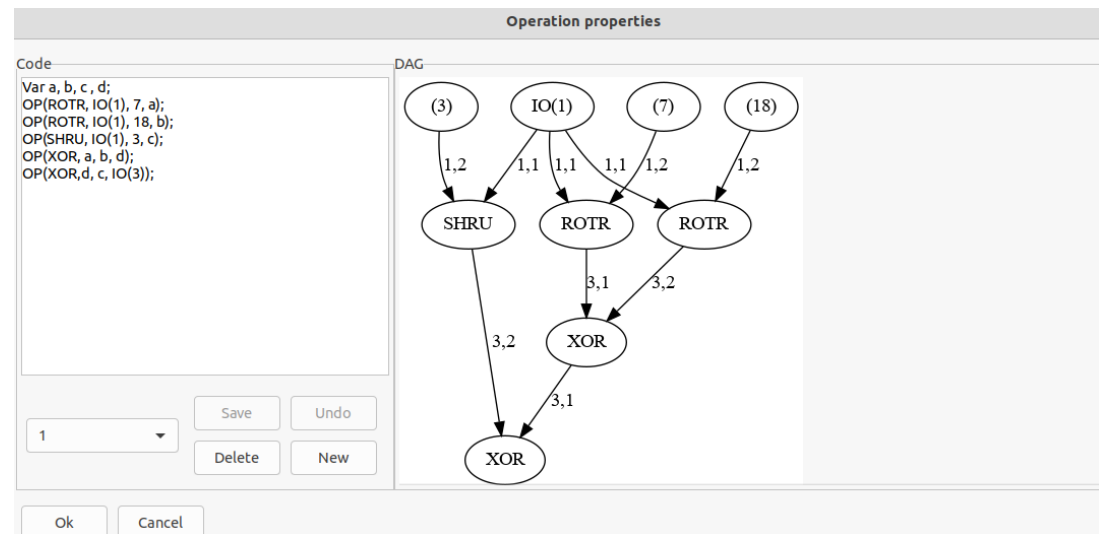
Operation outputs

operand	type	element width	element
3	SintWord	32	1

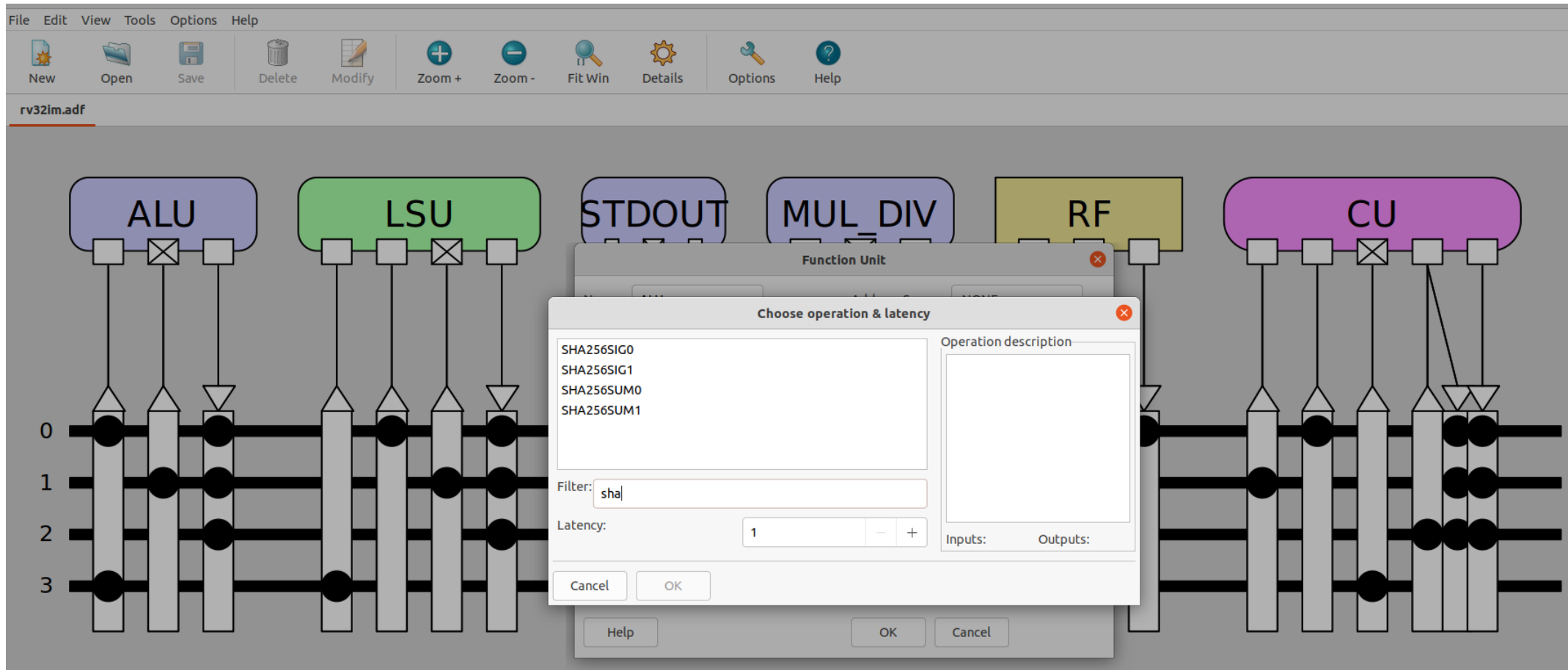
Add... Modify... Delete

Operation behavior module not defined. Open **Open DAG** OK Cancel

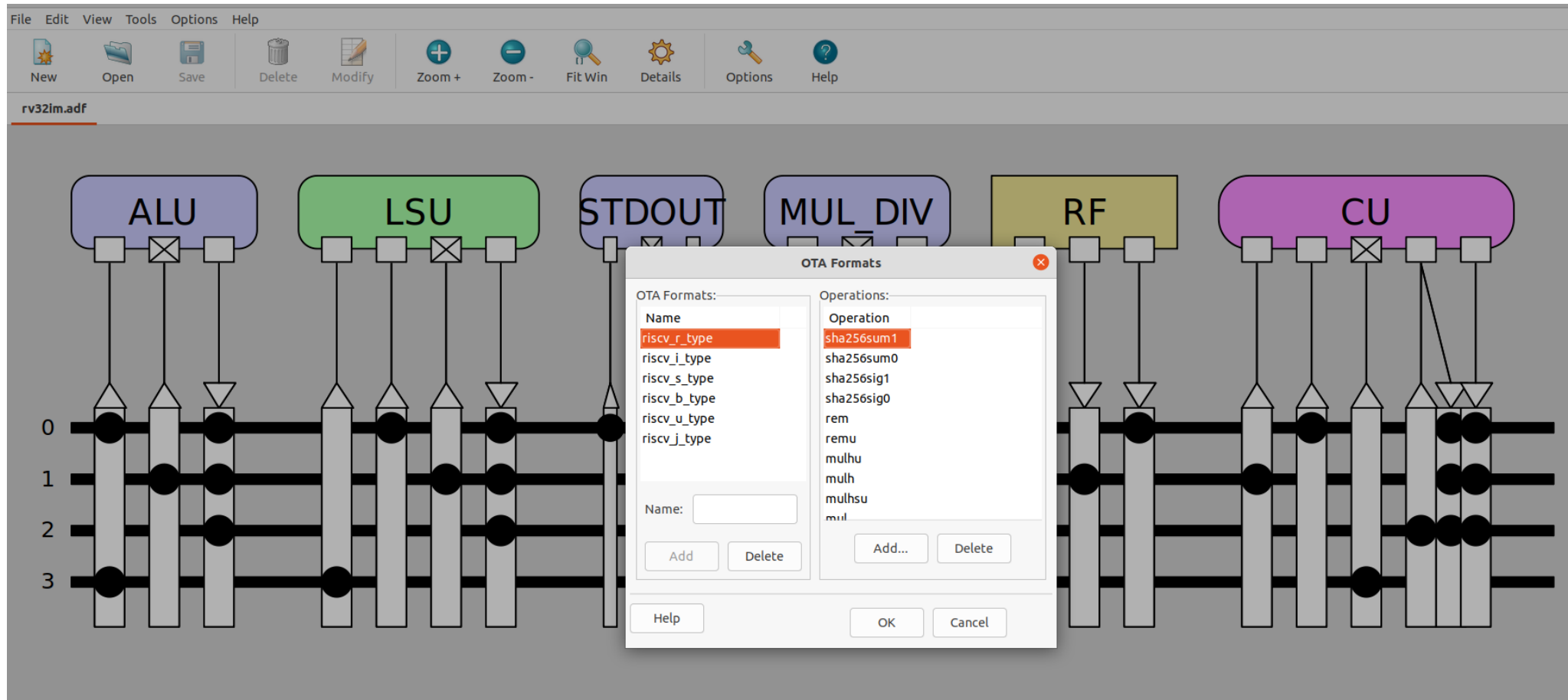
## Add a DAG-description to the entry



# 3. Modifying the Processor Description



# 3. Modifying the Processor Description



## 4. Modify the Source Code

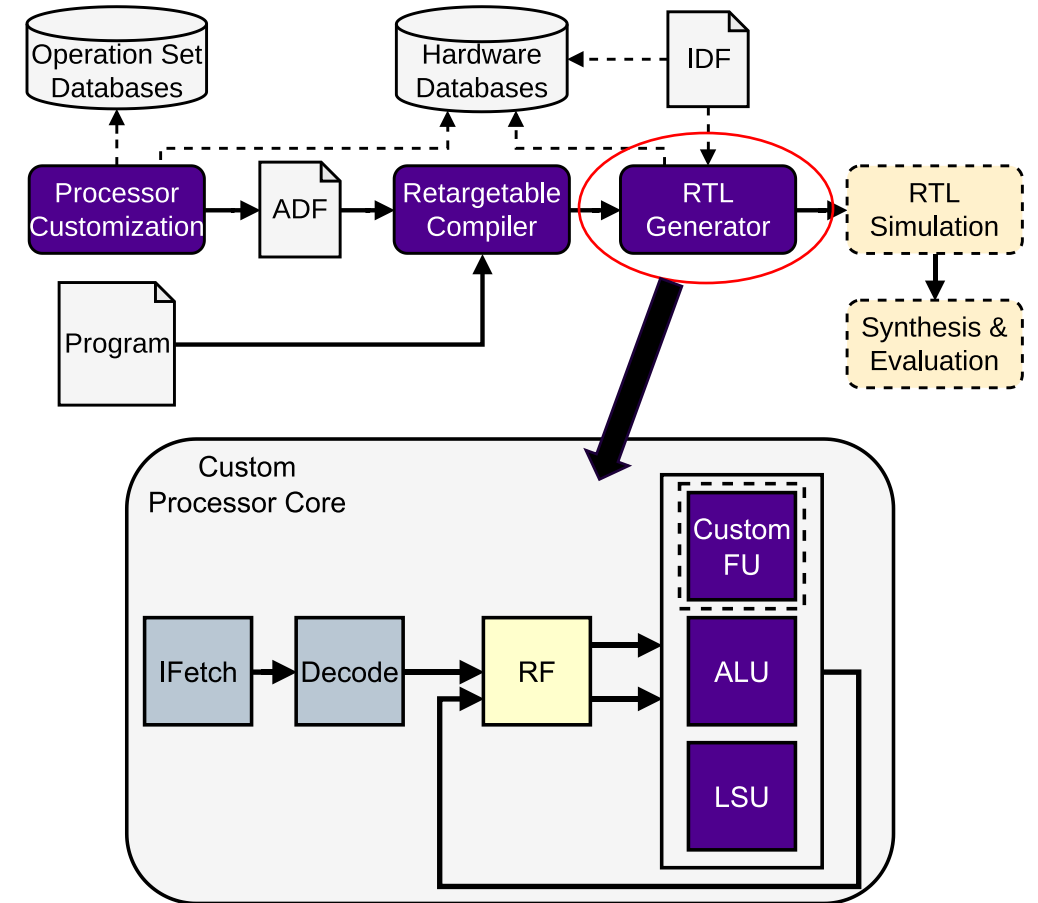
- Add the intrinsic calls
  - Definitions automatically generated by the compiler
- Automatic instruction selection:
  - Sometimes not feasible
  - Long operation chains
  - Compiler backend generation
  - Not currently supported for RISC-V

```
#define ROTL32(n,x) (((x)<<(n)) | ((x)>>((-n)&31)))

inline uint32_t S0(const uint32_t x) {
    uint32_t val = 0;
    #ifdef CUSTOM_ISE
        _OA_RV_SHA256SUM0(x, 0, val);
    #else
        val = ROTL32(30,(x)) ^ ROTL32(19,(x)) ^ ROTL32(10,(x));
    #endif
    return val;
}
```

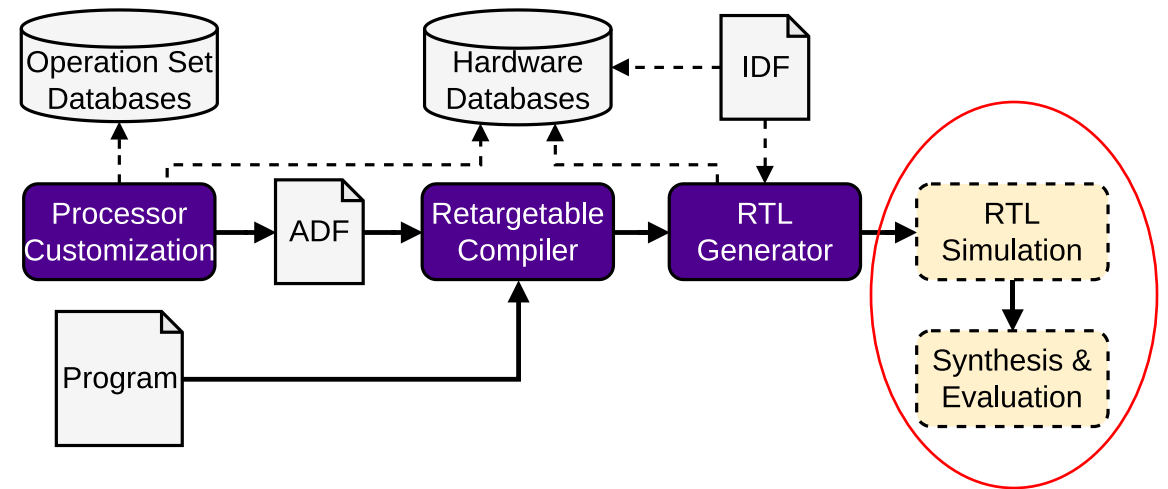
# Hardware Generation

- Command line tool
- Implementation definition file
  - References FUs and RFs
  - Optional
- Picks RFs and FUs from hardware databases
  - Generates FUs if not available
  - FUGen
- Decoder and datapath generation



# Results

- Synthesized with a 28 nm ASIC technology.
  - 49% reduction in run time
  - Negligible 1.5% area overhead
  - 1.4% reduction in clock frequency
- Rapid ASIP design without any RTL written by the designer.



# Future Work

- Integration of the hardware generation with CVX-IF
  - Enables to generate instruction set extensions to any RISC-V processor implementing CVX-IF
  - TRISTAN WI2.5.6
- Backend generation
  - Automatic instruction selection



# Conclusions

- HW/SW co-design toolset for RISC-V ASIPs
  - Generate and program customized RISC-V processors
  - Retargetable compiler
  - Hardware generation down to RTL
- Modular components
  - Operation set libraries
  - Hardware databases
  - Architecture definition file
- Processor customization without using HDLs
  - DAG-based operation descriptions

# Thank you!

**Kari Hepola**, Joonas Multanen and Pekka Jääskeläinen  
Customized Parallel Computing (CPC) Group, Tampere University, Finland

<https://www.tuni.fi/cpc>

<https://github.com/cpc/openasip>