Setup

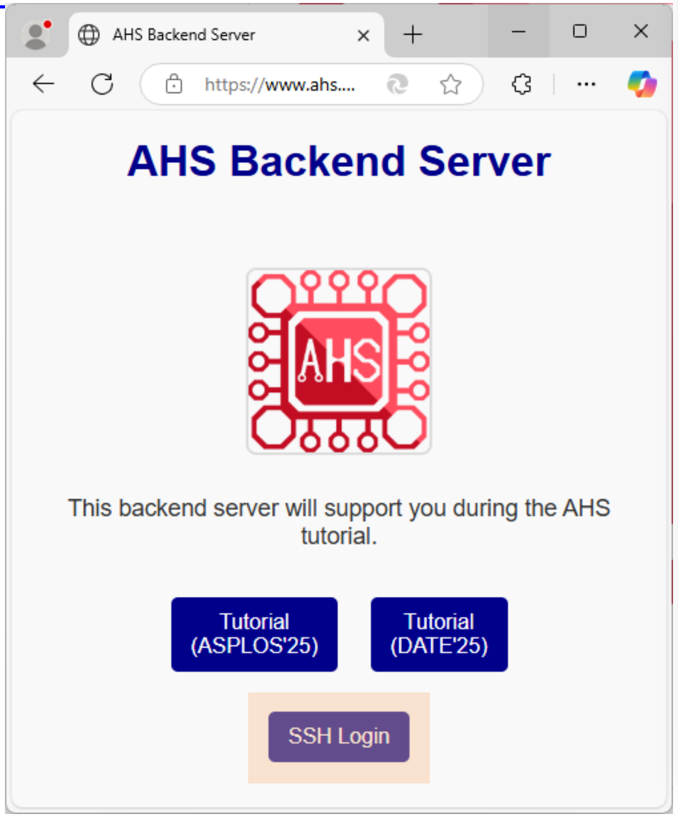
- A: Download the docker image
  - Please follow the handout or our website
  - https://ericlyun.me/tutorial-date2025
- B: Login our server
  - First, you need a SSH key
  - check if your SSH directory exists
    - win: C:\Users\<username>/.ssh
    - linux: $HOME/.ssh
    - mac: /Users/<username>/.ssh
  - if not exists, you need to create one
    - reference link

```
# Windows, open PowerShell
⌨  ssh-keygen -t ed25519 -C "<email>"
# Linux, open terminal
⌨  ssh-keygen -t ed25519 -C "<email>"
# Mac, open terminal
⌨  ssh-keygen -t ed25519 -C "<email>"
```

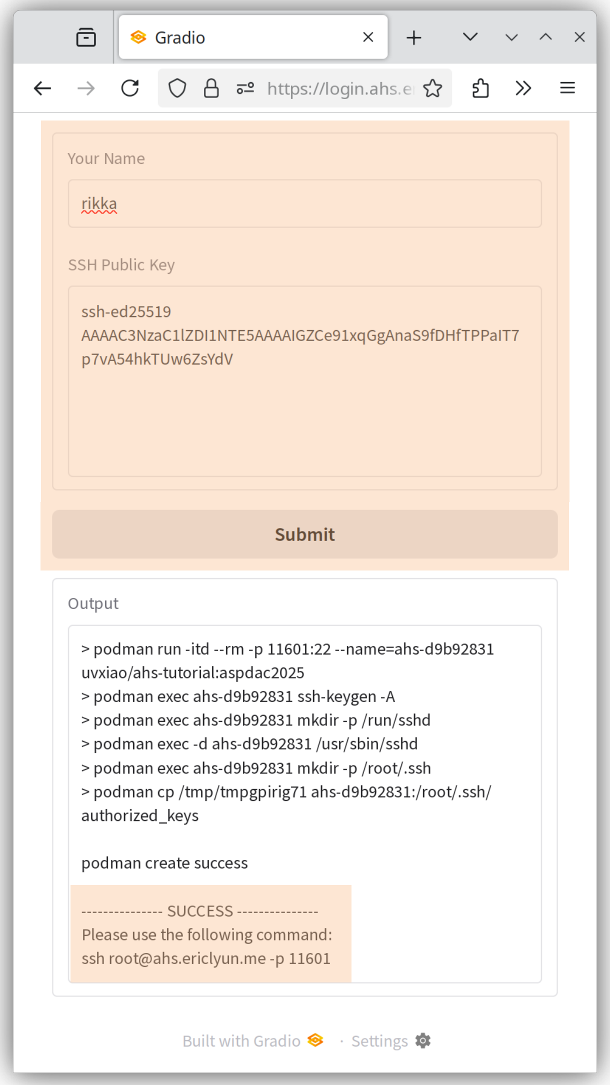
SSH public key can be found in <SSH>/id_ed25519.pub

Setup

- A: Download the docker image
  - Please follow the handout or our website
  - https://ericlyun.me/tutorial-date2025
- B: Login our server
  - www.ahs.ericlyun.me
  - Click SSH Login
  - Enter your name and ssh key
  - Follow the output to connect



This backend server will support you during the AHS tutorial.



Your Name: rikka

SSH Public Key:

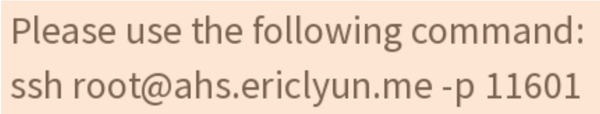
```
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIGZCe91xqGgAnaS9fDHfTPPaIT7
p7vA54hkTUw6ZsYdV
```

Output

```
> podman run -itd --rm -p 11601:22 --name=ahs-d9b92831
uvxiao/ahs-tutorial:aspdac2025
> podman exec ahs-d9b92831 ssh-keygen -A
> podman exec ahs-d9b92831 mkdir -p /run/sshd
> podman exec -d ahs-d9b92831 /usr/sbin/sshd
> podman exec ahs-d9b92831 mkdir -p /root/.ssh
> podman cp /tmp/tmpgpirig71 ahs-d9b92831:/root/.ssh/
authorized_keys

podman create success

--------------- SUCCESS ---------------
Please use the following command:
ssh root@ahs.ericlyun.me -p 11601
```

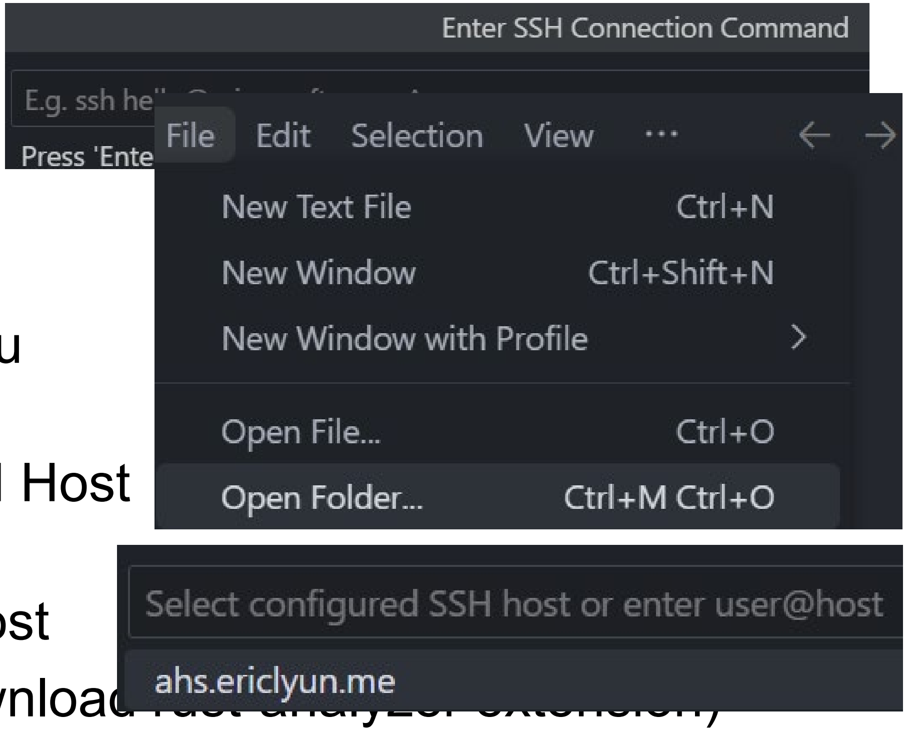
Setup

- A: Download the docker image
  - Please follow the handout or our website
  - https://ericlyun.me/tutorial-date2025



Please use the following command:
ssh root@ahs.ericlyun.me -p 11601

- B: Login our server
  - You can use PowerShell/terminal
  - Or, you can use VSCode
    - Ctrl-Shift-P / ⇧⌘P, type Remote: Show Remote Menu
      - select SSH to install the extension
    - Ctrl-Shift-P / ⇧⌘P, type Remote-SSH: Add New SSH Host
      - paste "ssh root@ahs.ericlyun.me –p xxxxx"
    - Ctrl-Shift-P / ⇧⌘P, type Remote-SSH: Connect to Host
    - Ctrl-K + Ctrl-O, open /root/repos folder (optional: download rust analyzer extension)

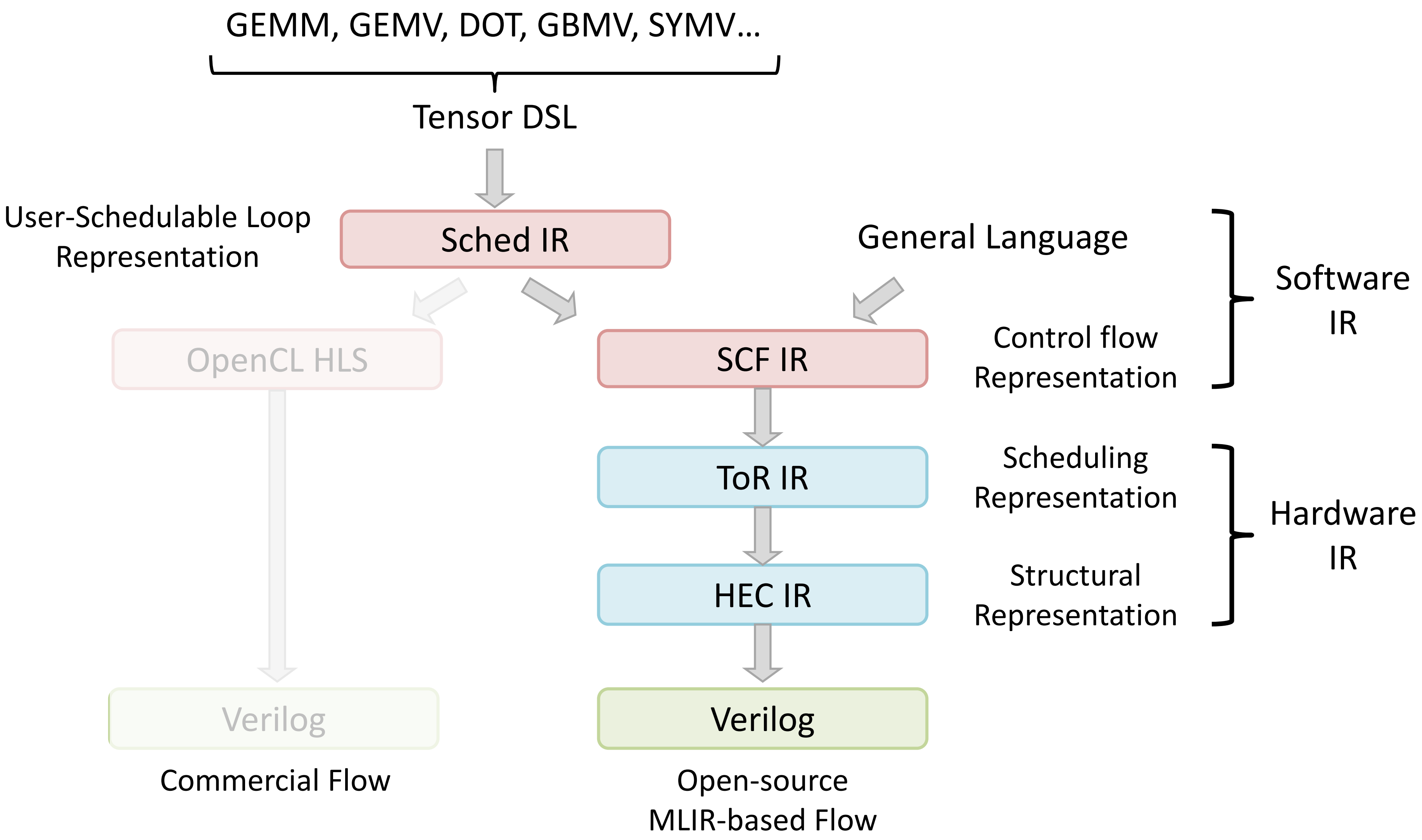


Outline

- High-level synthesis and DSL
  - Hector (ICCAD'22), FCCM'23, MICRO'24, TRETS'25

- HDL for Chip Design
  - Cement (FPGA'24)

Synthesis Flow

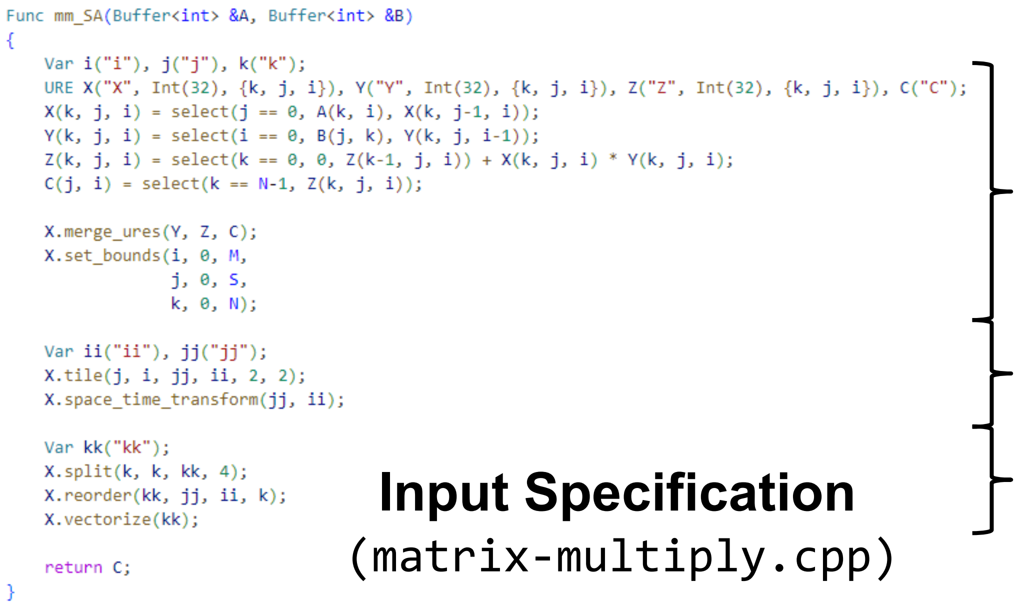

GEMM, GEMV, DOT, GBMV, SYMV…
Tensor DSL
User-Schedulable Loop Representation
Sched IR
General Language
Software IR
OpenCL HLS
SCF IR
Control flow Representation
ToR IR
Scheduling Representation
Hardware IR
HEC IR
Structural Representation
Verilog
Verilog
Commercial Flow
Open-source MLIR-based Flow

Quick Try

- **Run:**

```
cd popa/examples
bash ./hls-tutorial.sh generate [basic/SA]
```

- **Input and Output**



```
Func mm_SA(Buffer<int> &A, Buffer<int> &B)
{
    Var i("i"), j("j"), k("k");
    URE X("X", Int(32), {k, j, i}), Y("Y", Int(32), {k, j, i}), Z("Z", Int(32), {k, j, i}), C("C");
    X(k, j, i) = select(j == 0, A(k, i), X(k, j-1, i));
    Y(k, j, i) = select(i == 0, B(j, k), Y(k, j, i-1));
    Z(k, j, i) = select(k == 0, 0, Z(k-1, j, i)) + X(k, j, i) * Y(k, j, i);
    C(j, i) = select(k == N-1, Z(k, j, i));

    X.merge_ures(Y, Z, C);
    X.set_bounds(i, 0, M,
                 j, 0, S,
                 k, 0, N);

    Var ii("ii"), jj("jj");
    X.tile(j, i, jj, ii, 2, 2);
    X.space_time_transform(jj, ii);

    Var kk("kk");
    X.split(k, k, kk, 4);
    X.reorder(kk, jj, ii, k);
    X.vectorize(kk);

    return C;
}
```

Input Specification
`(matrix-multiply.cpp)`

Uniform Recurrence Equation (URE)

Space-time Transform (STT)

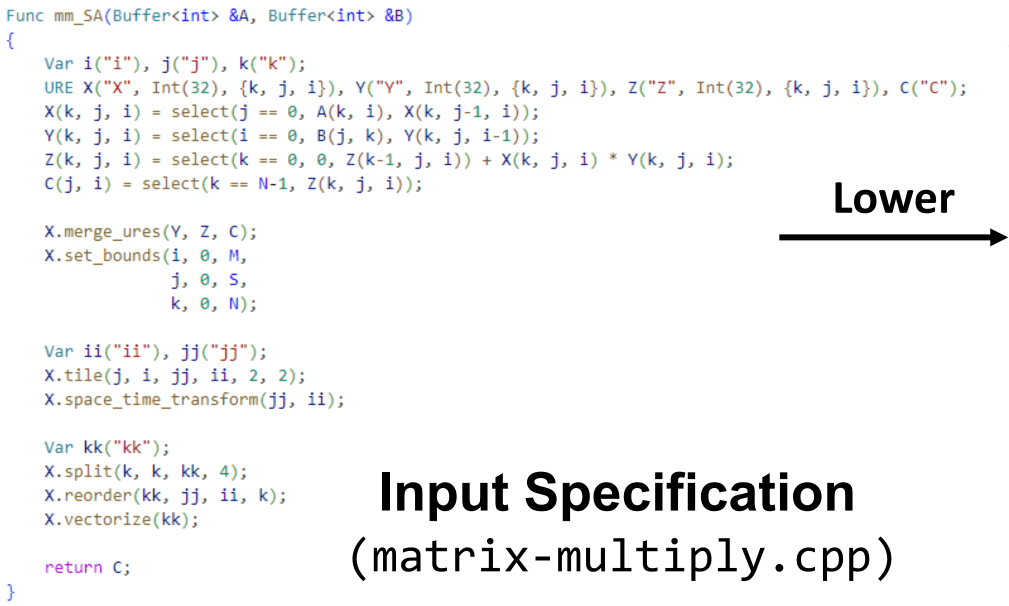
Loop Transformation

Quick Try

- **Run:**

```
cd popa/examples
bash ./hls-tutorial.sh generate [basic/SA]
```

- **Input and Output**

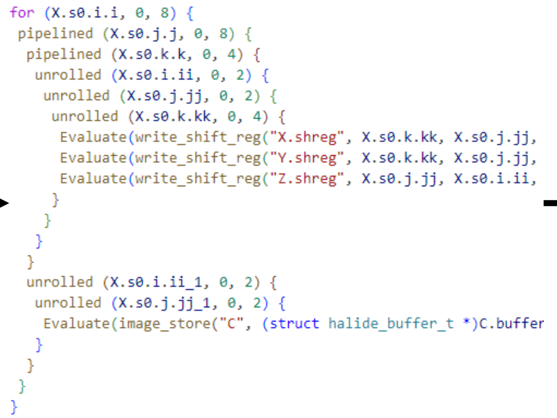


```
Func mm_SA(Buffer<int> &A, Buffer<int> &B)
{
    Var i("i"), j("j"), k("k");
    URE X("X", Int(32), {k, j, i}), Y("Y", Int(32), {k, j, i}), Z("Z", Int(32), {k, j, i}), C("C");
    X(k, j, i) = select(j == 0, A(k, i), X(k, j-1, i));
    Y(k, j, i) = select(i == 0, B(j, k), Y(k, j, i-1));
    Z(k, j, i) = select(k == 0, 0, Z(k-1, j, i)) + X(k, j, i) * Y(k, j, i);
    C(j, i) = select(k == N-1, Z(k, j, i));

    X.merge_ures(Y, Z, C);
    X.set_bounds(i, 0, M,
                 j, 0, S,
                 k, 0, N);

    Var ii("ii"), jj("jj");
    X.tile(j, i, jj, ii, 2, 2);
    X.space_time_transform(jj, ii);

    Var kk("kk");
    X.split(k, k, kk, 4);
    X.reorder(kk, jj, ii, k);
    X.vectorize(kk);

    return C;
}
```

Input Specification

`(matrix-multiply.cpp)`

Lower

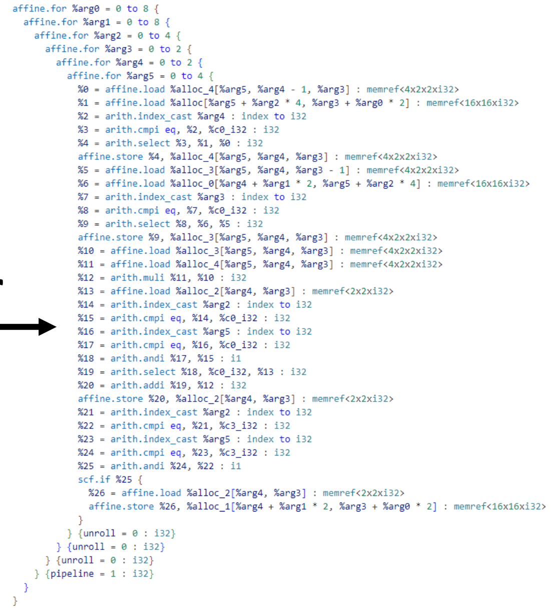


```
for (X.s0.i.i, 0, 8) {
 pipelined (X.s0.j.j, 0, 8) {
  pipelined (X.s0.k.k, 0, 4) {
   unrolled (X.s0.i.ii, 0, 2) {
    unrolled (X.s0.j.jj, 0, 2) {
     unrolled (X.s0.k.kk, 0, 4) {
      Evaluate(write_shift_reg("X.shreg", X.s0.k.kk, X.s0.j.jj,
      Evaluate(write_shift_reg("Y.shreg", X.s0.k.kk, X.s0.j.jj,
      Evaluate(write_shift_reg("Z.shreg", X.s0.j.jj, X.s0.i.ii,
     }
    }
   }
  }
  unrolled (X.s0.i.ii_1, 0, 2) {
   unrolled (X.s0.j.jj_1, 0, 2) {
    Evaluate(image_store("C", (struct halide_buffer_t *)C.buffer
   }
  }
 }
}
```

Sched IR

`(SchedIR_SA)`

Lower



```
affine.for %arg0 = 0 to 8 {
  affine.for %arg1 = 0 to 8 {
    affine.for %arg2 = 0 to 4 {
      affine.for %arg3 = 0 to 2 {
        affine.for %arg4 = 0 to 2 {
          affine.for %arg5 = 0 to 4 {
            %0 = affine.load %alloc_4[%arg5, %arg4 - 1, %arg3] : memref<4x2x2xi32>
            %1 = affine.load %alloc[%arg5 + %arg2 * 4, %arg3 + %arg0 * 2] : memref<16x16xi32>
            %2 = arith.index_cast %arg4 : index to i32
            %3 = arith.cmpi eq, %2, %c0_i32 : i32
            %4 = arith.select %3, %1, %0 : i32
            affine.store %4, %alloc_4[%arg5, %arg4, %arg3] : memref<4x2x2xi32>
            %5 = affine.load %alloc_3[%arg5, %arg4, %arg3 - 1] : memref<4x2x2xi32>
            %6 = affine.load %alloc_0[%arg4 + %arg1 * 2, %arg5 + %arg2 * 4] : memref<16x16xi32>
            %7 = arith.index_cast %arg3 : index to i32
            %8 = arith.cmpi eq, %7, %c0_i32 : i32
            %9 = arith.select %8, %6, %5 : i32
            affine.store %9, %alloc_3[%arg5, %arg4, %arg3] : memref<4x2x2xi32>
            %10 = affine.load %alloc_3[%arg5, %arg4, %arg3] : memref<4x2x2xi32>
            %11 = affine.load %alloc_4[%arg5, %arg4, %arg3] : memref<4x2x2xi32>
            %12 = arith.muli %11, %10 : i32
            %13 = affine.load %alloc_2[%arg4, %arg3] : memref<2x2xi32>
            %14 = arith.index_cast %arg2 : index to i32
            %15 = arith.cmpi eq, %14, %c0_i32 : i32
            %16 = arith.index_cast %arg5 : index to i32
            %17 = arith.cmpi eq, %16, %c0_i32 : i32
            %18 = arith.andi %17, %15 : i1
            %19 = arith.select %18, %c0_i32, %13 : i32
            %20 = arith.addi %19, %12 : i32
            affine.store %20, %alloc_2[%arg4, %arg3] : memref<2x2xi32>
            %21 = arith.index_cast %arg2 : index to i32
            %22 = arith.cmpi eq, %21, %c3_i32 : i32
            %23 = arith.index_cast %arg5 : index to i32
            %24 = arith.cmpi eq, %23, %c3_i32 : i32
            %25 = arith.andi %24, %22 : i1
            scf.if %25 {
              %26 = affine.load %alloc_2[%arg4, %arg3] : memref<2x2xi32>
              affine.store %26, %alloc_1[%arg4 + %arg1 * 2, %arg3 + %arg0 * 2] : memref<16x16xi32>
            }
          } {unroll = 0 : i32}
        } {unroll = 0 : i32}
      } {unroll = 0 : i32}
    } {pipeline = 1 : i32}
  }
}
```

SCF IR

`(SCF_SA.mlir)`

How to Express the Dataflow?

Each loop iteration is associated with unique variables: $X_{ijk}, Y_{ijk}, Z_{ijk}$

$$\text{UREs} \begin{cases} X_{ijk} = a_{ik} \text{ if } j=0,\ X_{i(j\text{-}1)k} \text{ otherwise} \\ Y_{ijk} = b_{kj} \text{ if } i=0,\ Y_{(i\text{-}1)jk} \text{ otherwise} \\ Z_{ijk} = 0 \text{ if } k=0,\ Z_{ij(k\text{-}1)} + X_{ijk}\, Y_{ijk} \text{ otherwise} \end{cases}$$

Final result: Z_{ijk} if k reaches its max

Uniform Recurrence Equations (UREs):

- Recursive definition
- Uniform dependency
- Dynamic single assignment form

Specification

```
X(k, j, i) = select(j == 0, A(k, i), X(k, j-1, i));
Y(k, j, i) = select(i == 0, B(j, k), Y(k, j, i-1));
Z(k, j, i) = select(k == 0, 0, Z(k-1, j, i)) + X(k, j, i) * Y(k, j, i);
C(j, i) = select(k == K-1, Z(k, j, i));
```

select(condition, x, y)
= (condition ? x : y)

Dataflow Mapping

- **Space-time Transform**
  - Mapping loop instances onto spatial PEs
  - Schedule instance execution in a temporal order

Map each point to a new space of $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{t})$

$$STT\left(\begin{bmatrix} i \\ j \\ k \end{bmatrix}\right) = \begin{bmatrix} x \\ y \\ t \end{bmatrix} \begin{matrix} \}\text{ Space} \\ \}\text{ Time} \end{matrix} \qquad STT = \begin{pmatrix} P \\ T \end{pmatrix}$$

Space part:

$$P = \begin{array}{c} \begin{array}{ccc} i & j & k \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \quad \begin{array}{l} x = i \\ y = j \end{array}$$

Map point (i, j, k) to (x, y)

Time part:

$$\mathrm{T} = \begin{array}{c} \begin{array}{ccc} i & j & k \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \end{array} \quad t = k + j + i$$

Schedule point (i, j, k) at time t

Dataflow Mapping

- **Space-time Transform**
  - Mapping loop instances onto spatial PEs
  - Schedule instance execution in a temporal order

PE(0,0) t=0:

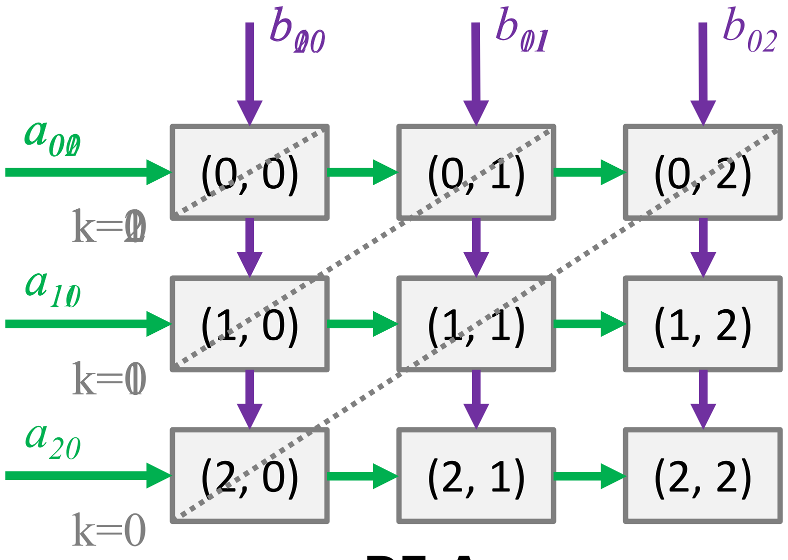
Map a_{00}->X_{000}, b_{00}->Y_{000}, 0->Z_{000}

PE(0,0) t=1:

Map a_{01}->X_{001}, b_{10}->Y_{001}, Z_{000}->Z_{001}
Map X_{000}->X_{010} (reuse of a_{00})
Map Y_{000}->Y_{100} (reuse of b_{00})

PE(0,0) t=2:

Map a_{02}->X_{002}, b_{20}->Y_{002}, Z_{001}->Z_{002}
Map X_{001}->X_{011} (reuse of a_{01})
Map Y_{001}->Y_{101} (reuse of b_{10})



PE Array

Quick Try

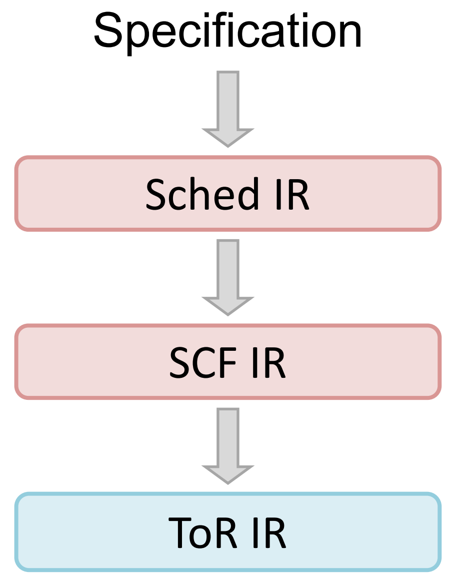
- **Run:**

```
./hls-tutorial run basic
./hls-tutorial run SA
```

- **Then you will get:**

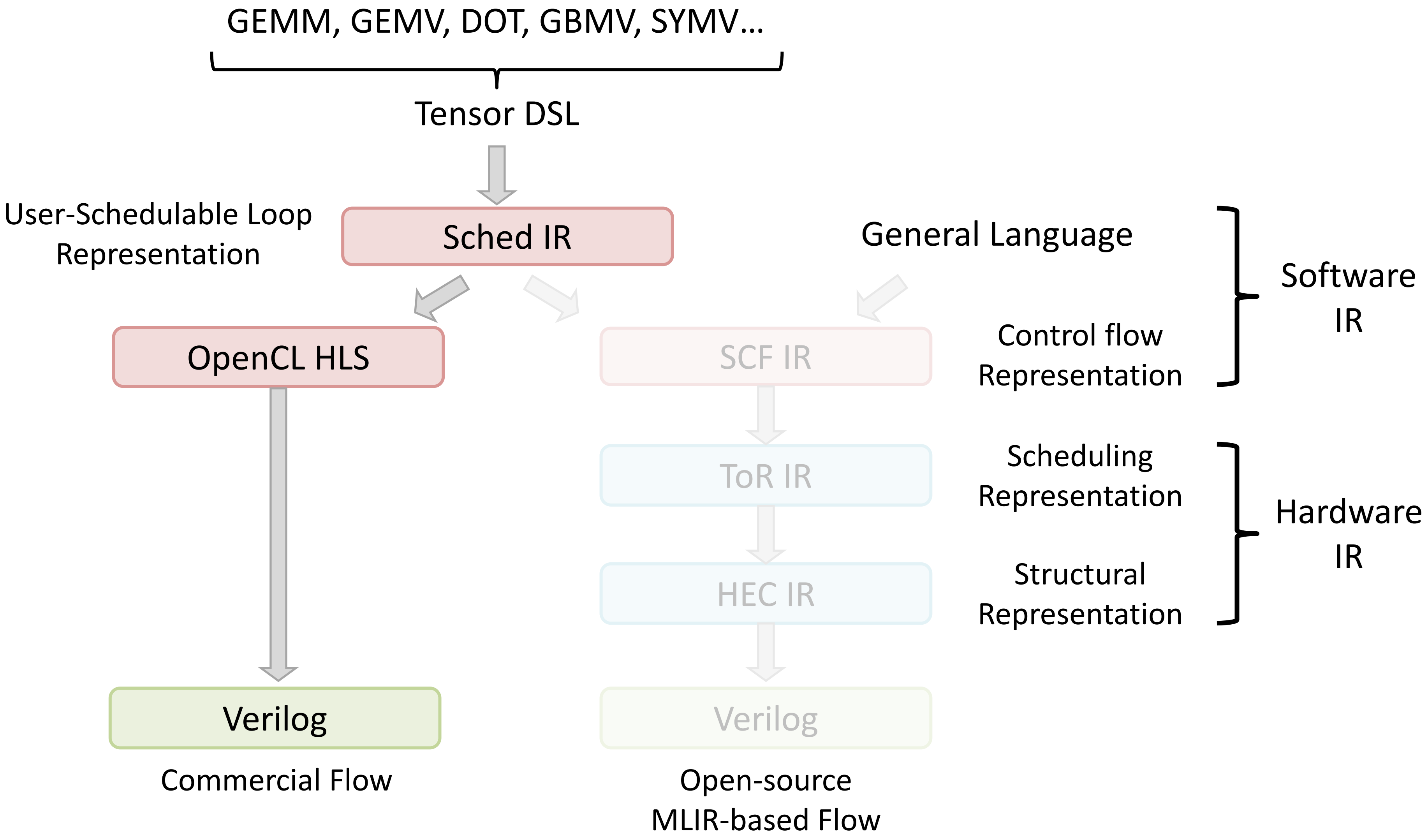
	basic	SA
Cycle	15411	1691

The systolic array achieves a 9X speedup compared to the basic version



Interpreted execution at the ToR IR level

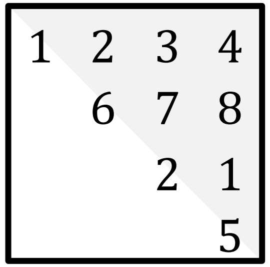
Synthesis Flow


GEMM, GEMV, DOT, GBMV, SYMV…
Tensor DSL
User-Schedulable Loop Representation
Sched IR
General Language
Software IR
OpenCL HLS
SCF IR
Control flow Representation
ToR IR
Scheduling Representation
Hardware IR
HEC IR
Structural Representation
Verilog
Verilog
Commercial Flow
Open-source MLIR-based Flow

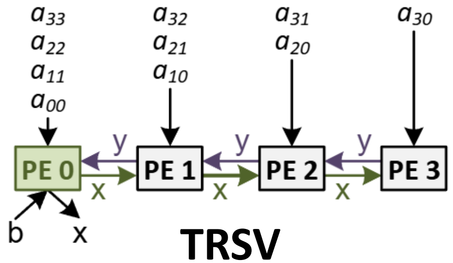
BLAS Development

- **Using the DSL to develop BLAS routines**
  - Navigate to `popa/examples/BLAS`

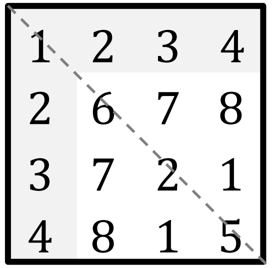
- **Leveraging the triangularity, symmetry, and band structure of matrices**
  - **TRSV:** Triangular matrix-vector solve
  - **SYMV:** Symmetric matrix-vector multiplication
  - **GBMV:** Banded matrix-vector multiplication



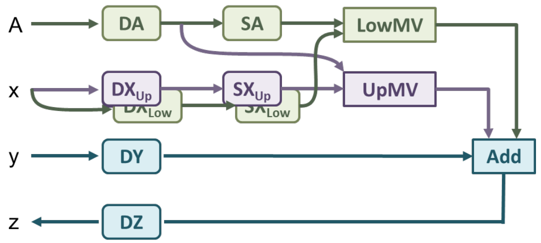
Triangular



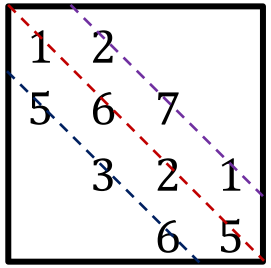
TRSV



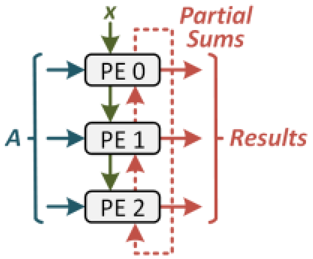
Symmetric



SYMV



Banded



GBMV

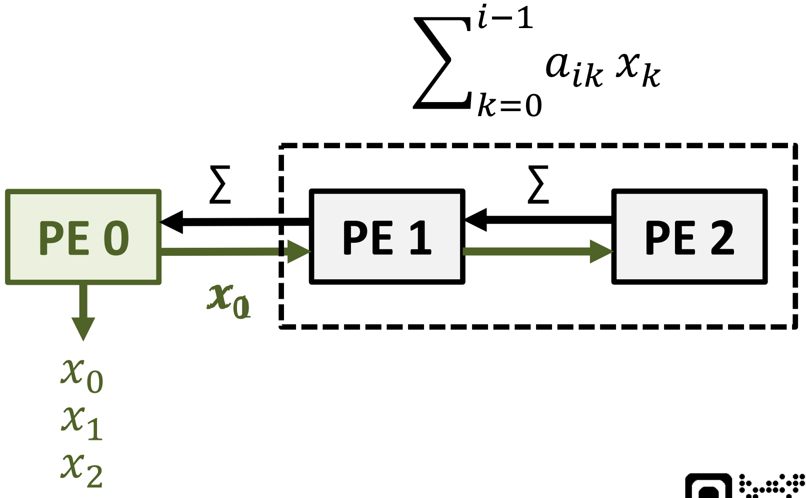
Example: TRSV

- **TRSV: Solve Ax=b where A is a triangular matrix**
  - A forward substitution process

$$\begin{array}{l} \text{PE 0} \\ \text{PE 1} \\ \text{PE 2} \end{array} \begin{bmatrix} a_{00} & & \\ a_{10} & a_{11} & \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

$$x_0 = b_0/a_{00}$$

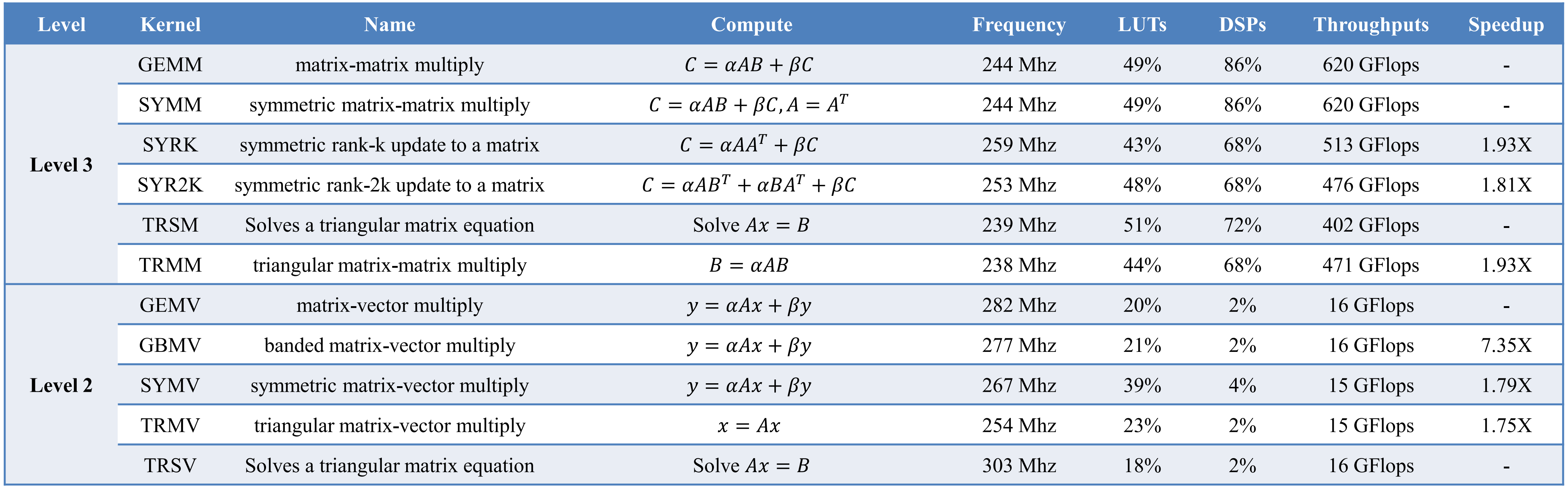
$$x_i = (b_i - \sum_{k=0}^{i-1} a_{ik}\, x_k)/a_{ii}$$



Our recent work - A dataflow accelerator for SpTRSV that exploits structural sparsity patterns in solving PDEs (ISCA'25)

github.com/pku-liang/telos

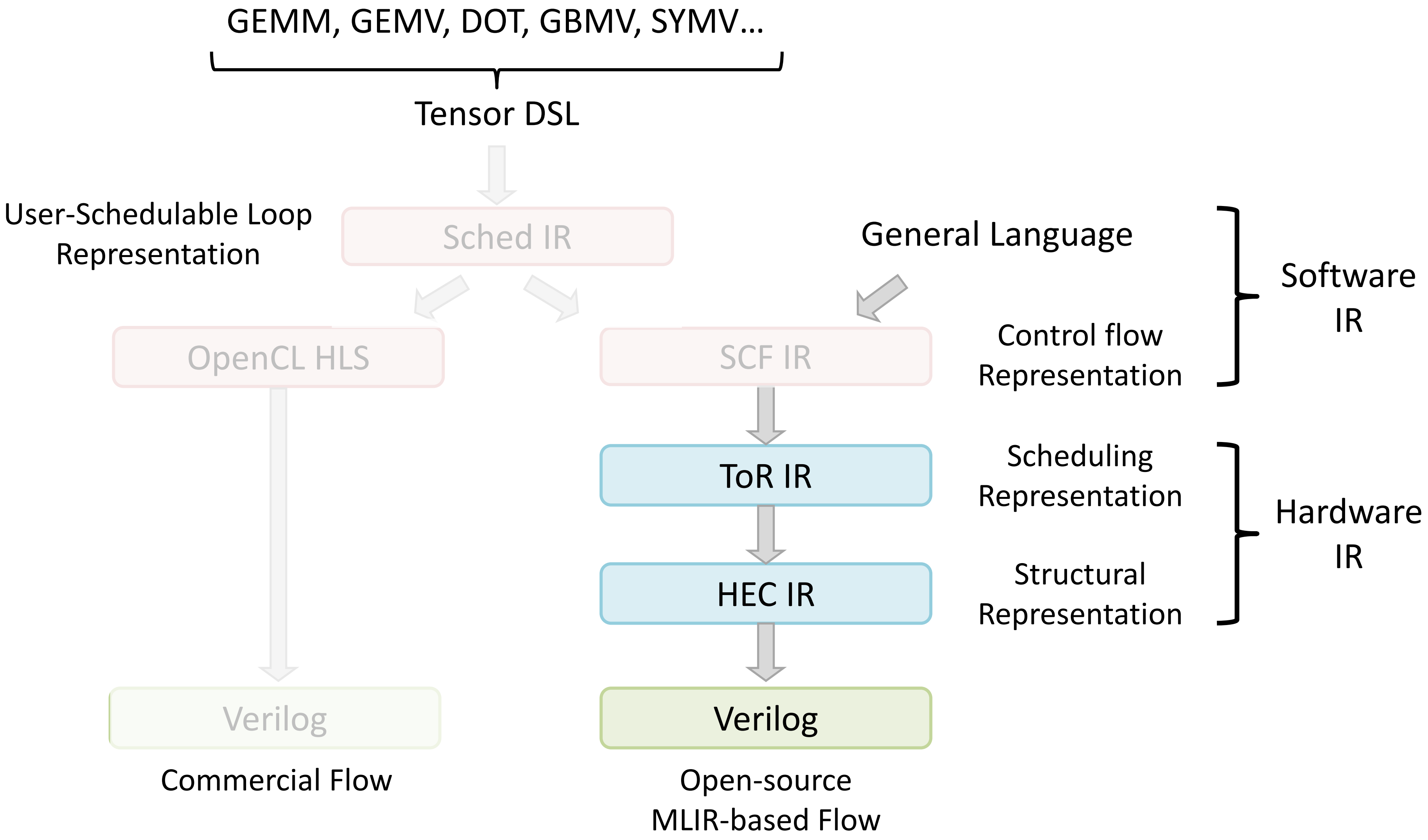
Results



Level	Kernel	Name	Compute	Frequency	LUTs	DSPs	Throughputs	Speedup
Level 3	GEMM	matrix-matrix multiply	$C = \alpha AB + \beta C$	244 Mhz	49%	86%	620 GFlops	-
	SYMM	symmetric matrix-matrix multiply	$C = \alpha AB + \beta C, A = A^T$	244 Mhz	49%	86%	620 GFlops	-
	SYRK	symmetric rank-k update to a matrix	$C = \alpha AA^T + \beta C$	259 Mhz	43%	68%	513 GFlops	1.93X
	SYR2K	symmetric rank-2k update to a matrix	$C = \alpha AB^T + \alpha BA^T + \beta C$	253 Mhz	48%	68%	476 GFlops	1.81X
	TRSM	Solves a triangular matrix equation	Solve $Ax = B$	239 Mhz	51%	72%	402 GFlops	-
	TRMM	triangular matrix-matrix multiply	$B = \alpha AB$	238 Mhz	44%	68%	471 GFlops	1.93X
Level 2	GEMV	matrix-vector multiply	$y = \alpha Ax + \beta y$	282 Mhz	20%	2%	16 GFlops	-
	GBMV	banded matrix-vector multiply	$y = \alpha Ax + \beta y$	277 Mhz	21%	2%	16 GFlops	7.35X
	SYMV	symmetric matrix-vector multiply	$y = \alpha Ax + \beta y$	267 Mhz	39%	4%	15 GFlops	1.79X
	TRMV	triangular matrix-vector multiply	$x = Ax$	254 Mhz	23%	2%	15 GFlops	1.75X
	TRSV	Solves a triangular matrix equation	Solve $Ax = B$	303 Mhz	18%	2%	16 GFlops	-

- Synthesized using Intel OpenCL SDK, tested on Intel A10 FPGA
- 1.8X-7.4X speedup by leveraging special matrix properties

Synthesis Flow


GEMM, GEMV, DOT, GBMV, SYMV…
Tensor DSL
User-Schedulable Loop Representation
Sched IR
General Language
Software IR
OpenCL HLS
SCF IR
Control flow Representation
ToR IR
Scheduling Representation
Hardware IR
HEC IR
Structural Representation
Verilog
Verilog
Commercial Flow
Open-source MLIR-based Flow

Quick Try

- **Run:**

```
cd hector
bash examples/hls_script.sh hector examples
```

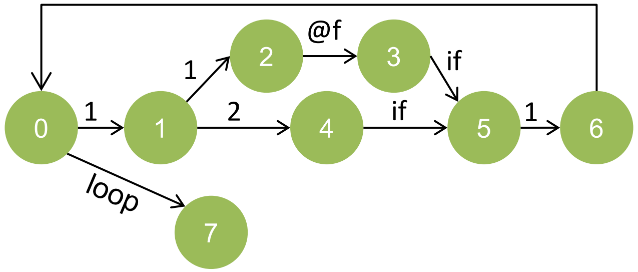
- **Then you will get several directories:**

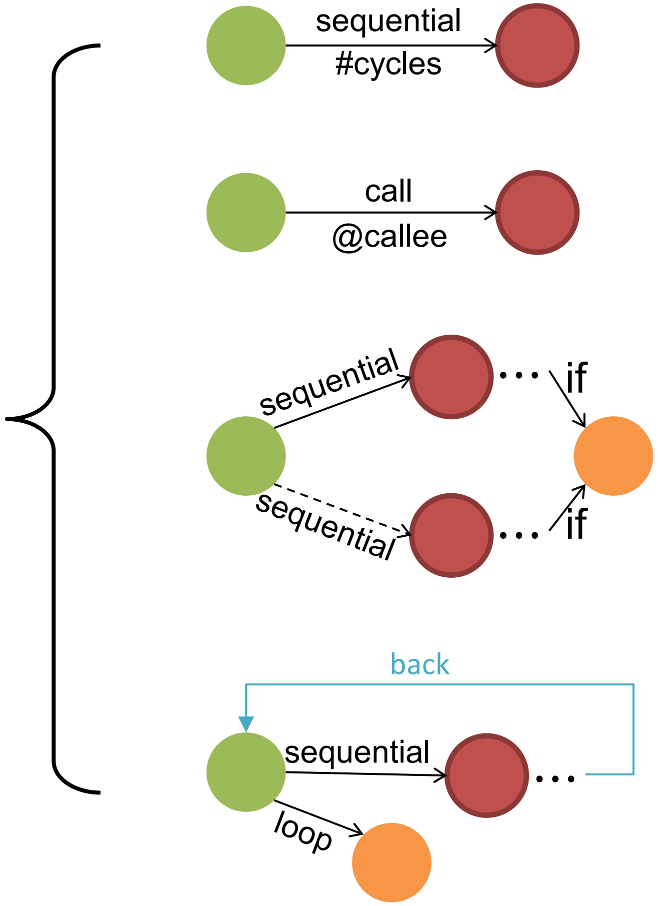
```
examples/tor-raw/
examples/tor-split/
examples/hec/
examples/chisel/
```

ToR IR

- **Provide a high-level control flow for hardware information**
  - Two parts, time graph and functional operations
  - Four types of nodes: normal, call, if and loop

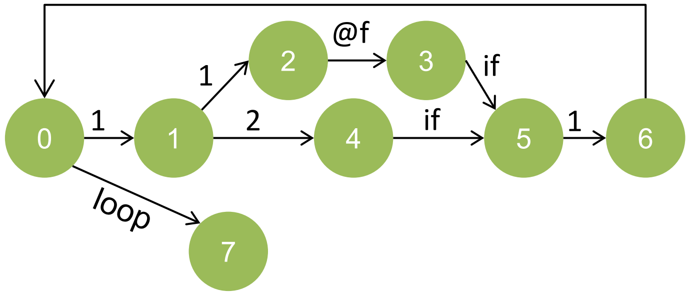
```
tor.topo (0 to 7) {
  tor.from 0 to 1 "seq:1"
  tor.from 1 to 2 "seq:1"
  tor.from 2 to 3 "call"
  tor.from 1 to 4 "seq:2"
  tor.from 3, 4 to 5 "if"
  tor.from 5 to 6 "seq:1"
  tor.from 0 to 7 "for"
}
```





ToR IR

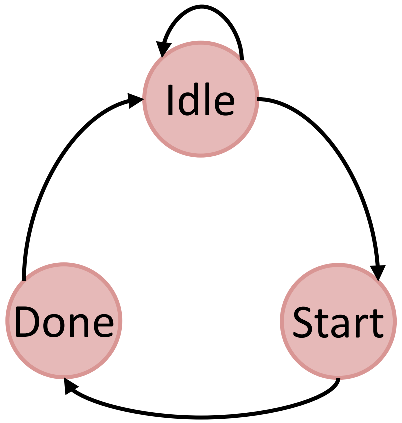
- **Functional operations present algorithmic specification with high-level information**
  - Bind each operation to the time graph



```
tor.for %i = %c0 to %c10 step %c1 {
  %m = tor.load %mask[%i] on (0 to 1)
  %a = tor.if %m then {
    %x = tor.addi %i %c1 on (1 to 2)
    %y = tor.subi %i %c1 on (1 to 2)
    %fx = tor.call @f(%x, %y) on (2 to 3)
    tor.yield %fx
  } else {
    %ii = tor.muli %i %i on (1 to 4)
    tor.yield %ii
  } on (1 to 5)
  tor.store %a to %A[%i] on (5 to 6)
} on (0 to 7)
```

HEC IR

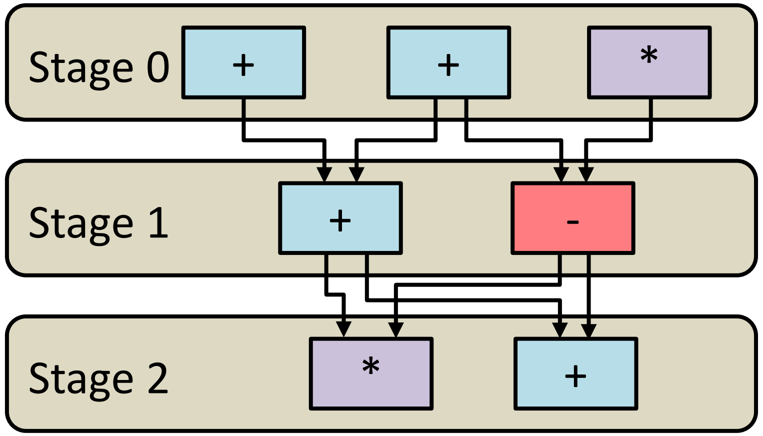
- **Three components**
  - State Transition Graph, Pipeline stages, Handshake



```
component @STG {
  // allocations
  stateset {
    state @Idle {
      // assigns
      transition {
        goto @Start if %cond
      }
    }
    state @Start {
      transition {
        goto @Done
      }
    }
    state @Done {
      transition {
        goto @Idle
      }
    }
  }
} {style = "stg"}
```

HEC IR

- **Three components**
  - State Transition Graph, Pipeline stages, Handshake

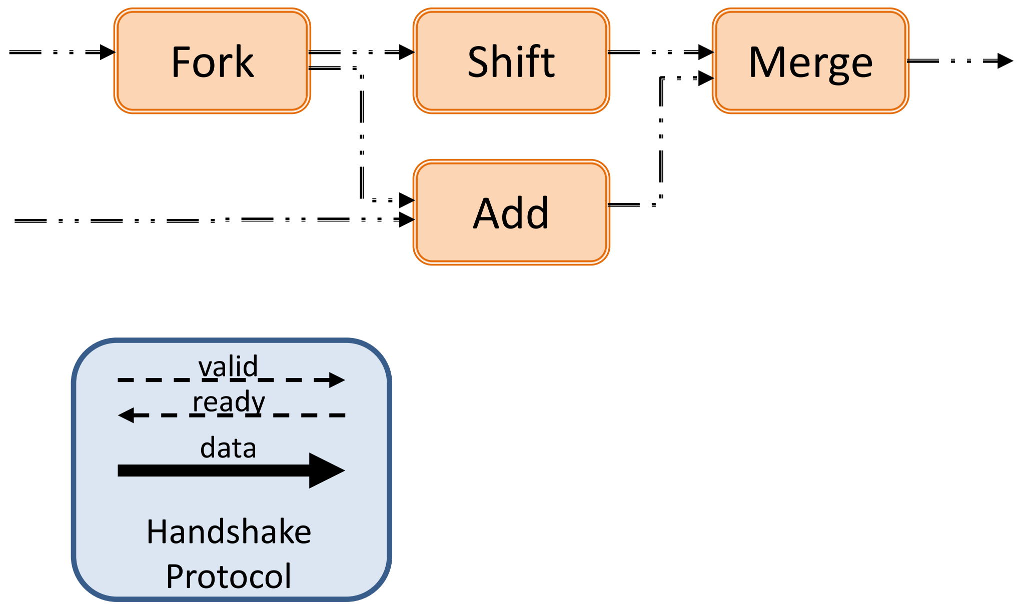


```
component @Pipe {
  // allocations
  stageset {
    stage @stage0 {
      // assigns
    }
    stage @stage1 {
      // assigns
      hec.assign %add2.lhs = %add0.res
      hec.assign %add2.rhs = %add1.res
    }
    stage @stage2 {
      // assigns
    }
    stage @stage3 {
      // assigns
      deliver %mul1.res, %add3.res
    }
  }
} {"pipeline",II=1}
```

HEC IR

- **Three components**
  - State Transition Graph, Pipeline stages, Handshake

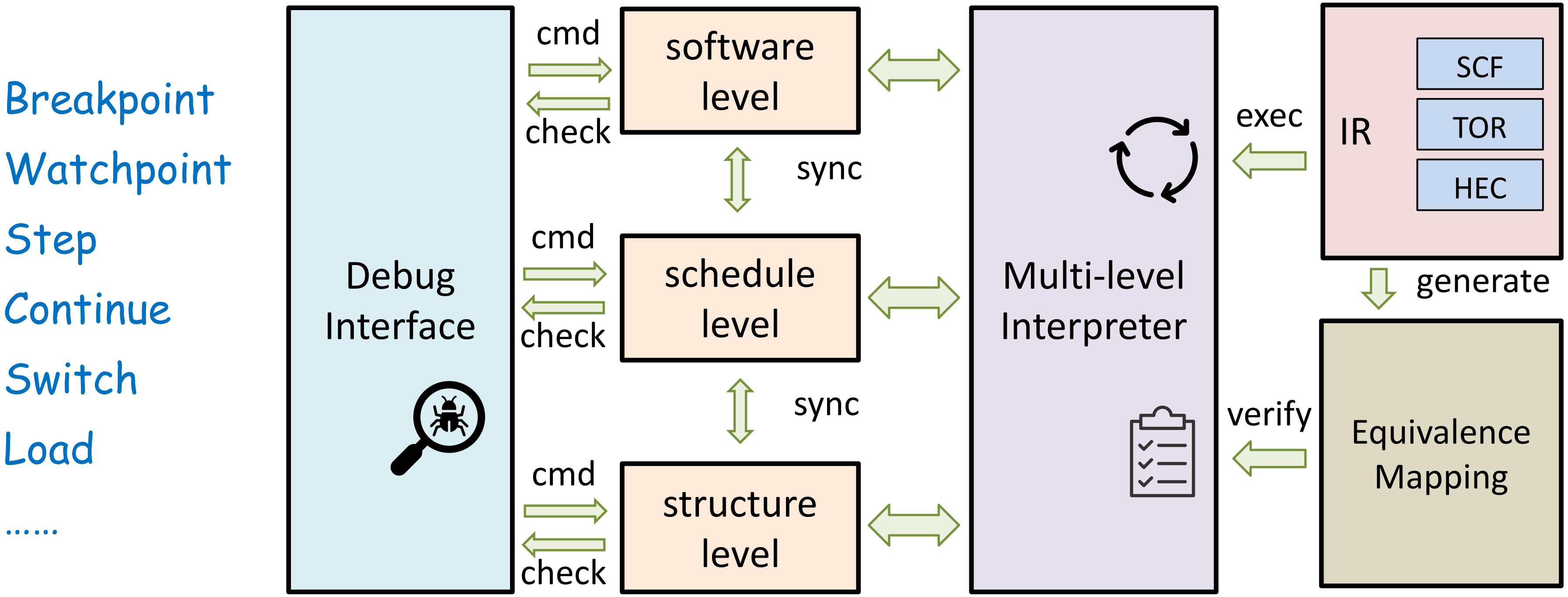
Latency-insensitive Design



```
component @Handshake (%in1,%in2,%out){
  // allocations
  %shift.in,res = primitive . "shift"
  %add.lhs,rhs,res = primitive . "addi"
  // elastic units
  %merge.i1,i2,o = primitive . "merge"
  %fork.i,o1,o2 = primitive . "fork"

  graph {
    assign %fork.i = %in1
    assign %shift.in = %fork.o1
    assign %add.lhs = %in2
    assign %add.rhs = %f.o2
    assign %merge.i1 = %shift.res
    assign %merge.i2 = %add.res
    assign %out  = %merge.o
  }
}{"handshake"}
```

Hestia

- An efficient cross-level debugger for HLS designs that enables breakpoints and stepping at multiple granularity



Debugging interface

- cd hestia/tutorial/examples/case1

- hestia command.tcl
- breakpoint op_116
- continue
- var op_115
- unset_breakpoint op_116
- watch op_135_b
- watch op_131_b
- watch op_115
- step 10000
- show_op
- step 2

```
SHOW VALUE:
        op_115 I32(0)
SHOW VALUE:
               op_115 I32(6)
        op_135_b I32(1)
        op_131_b I32(3)
        op_115 I32(6)
        op_135_b I32(1)
        op_131_b I32(3)
        op_115 I32(6) …
Computation { operands: ["op_135_b",
"op_113"], op_type: "shift_left", name:
"op_114", ret_type: "i32" }
SHOW VALUE:
        op_115 I32(6)
        op_135_b I32(1)
        op_131_b I32(3)
        op_115 I32(7)
        op_135_b I32(1)
        op_131_b I32(3)
```

Outline

- High-level synthesis and DSL
  - Hector (ICCAD'22), FCCM'23, MICRO'24, TRETS'25
- HDL for Chip Design
  - Cement (FPGA'24)

Cement: Evolved HDL and Framework for Chip Design

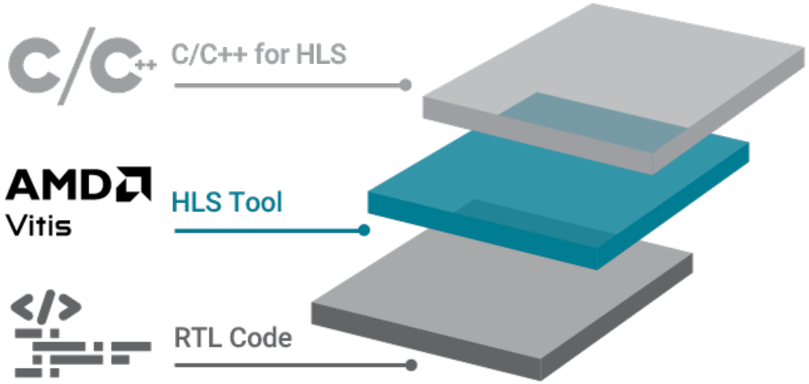
SystemVerilog

Traditional HDL
tedious hardware code

?

CHISEL

Embedded HDL (eHDL)
design hardware in advanced programming language (Scala, Python)



HLS: design hardware in C/C++

Rust-embedded HDL

software-like control description

Cement
(cmt2!)

keep evolving…

Cement (cmt1)

published at FPGA’24

Youwei Xiao, Zizhang Luo, Kexing Zhou, and Yun Liang. 2024. Cement: Streamlining FPGA Hardware Design with Cycle-Deterministic eHDL and Synthesis. FPGA '24. https://doi.org/10.1145/3626202.3637561

✚ *rule-based design*

bluespec™

High-level HDL: describe hardware as rules

Today, we introduce cmt2!

pku-liang / Cement Public

as the successor of …

Cement (cmt1)

published at FPGA’24

cmt2!

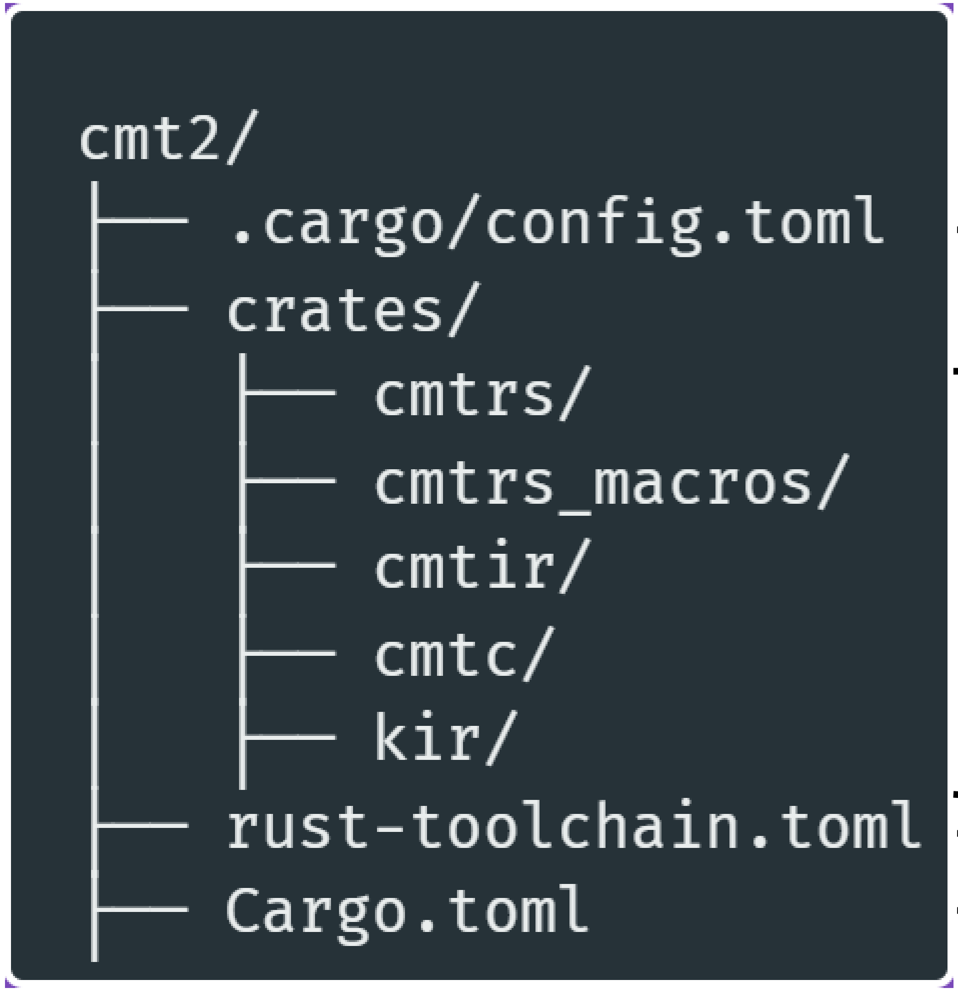
Rust-embedded HDL

rule-based design

software-like control description

In Rust, cmt2 project looks like…

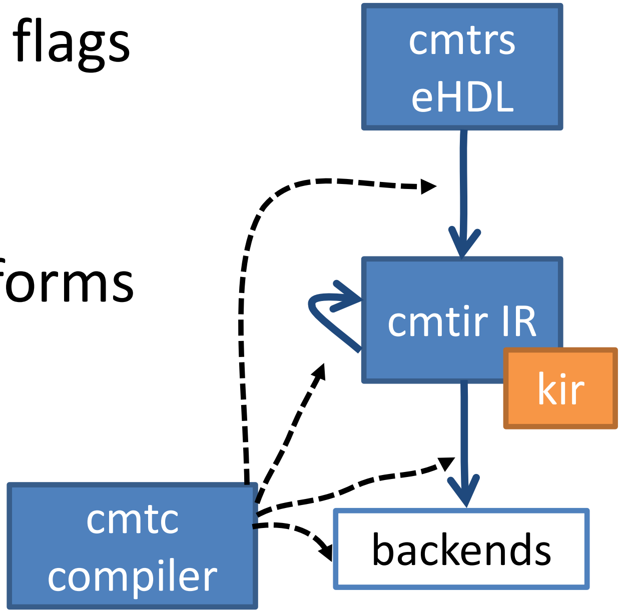
```
cd $REPOS/cmt2 && ls -a
```



```
cmt2/
├── .cargo/config.toml
├── crates/
│   ├── cmtrs/
│   ├── cmtrs_macros/
│   ├── cmtir/
│   ├── cmtc/
│   ├── kir/
├── rust-toolchain.toml
├── Cargo.toml
```

include necessary build flags

Here, 5 library *crates* forms the **cmt2** *package*

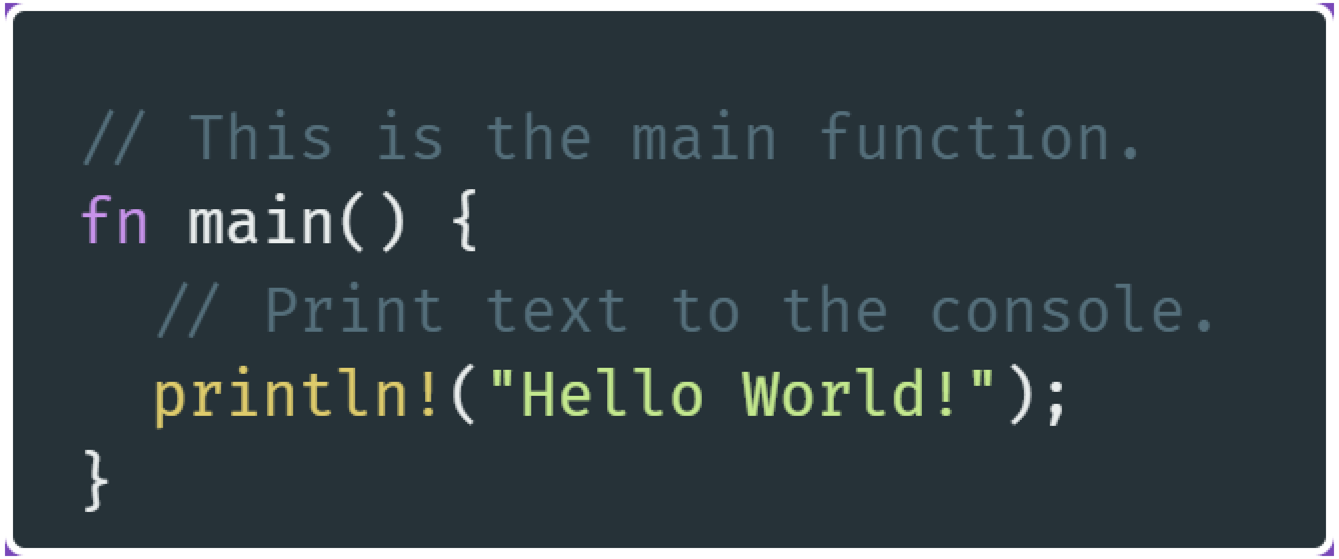


specify rust version

specify package information and dependencies

Hello Rust World!

```
⌨ touch
  $REPOS/cmt2/crates/cmtc/examples/hello.rs
```



```rust
// This is the main function.
fn main() {
  // Print text to the console.
  println!("Hello World!");
}
```

create `hello.rs` file ↑

click [run] above `fn main`, or ...

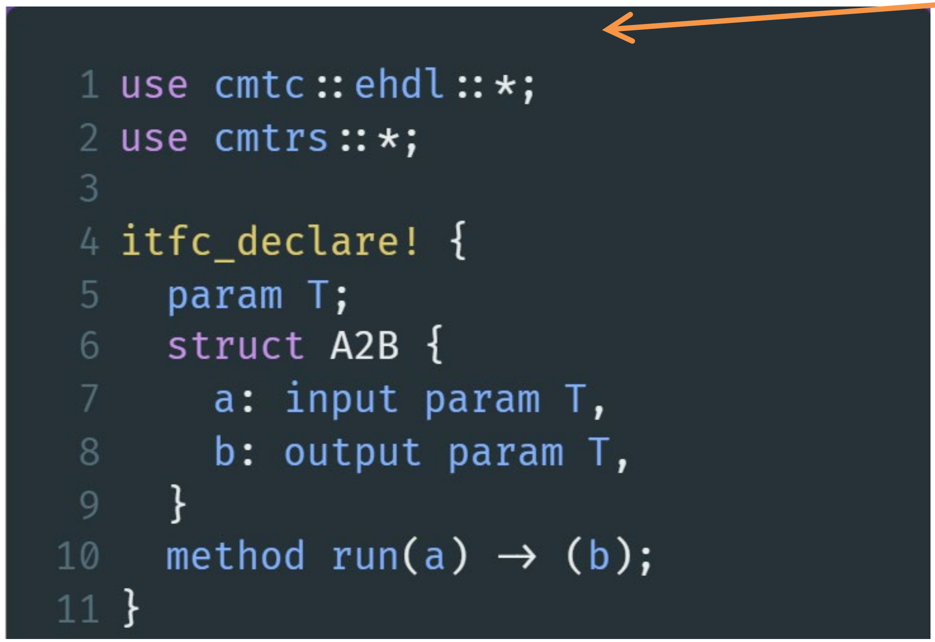
under directory `$REPOS/cmts`

```
⌨ cargo run --example hello
```

Cargo is Rust’s build system and package manager. It’s easy-to-use!
More user-friendly than sbt for Scala/Chisel!

Hello cmt2!

at the top of `hello.rs`



```
use cmtc::ehdl::*;
use cmtrs::*;

itfc_declare! {
  param T;
  struct A2B {
    a: input param T,
    b: output param T,
  }
  method run(a) -> (b);
}
```

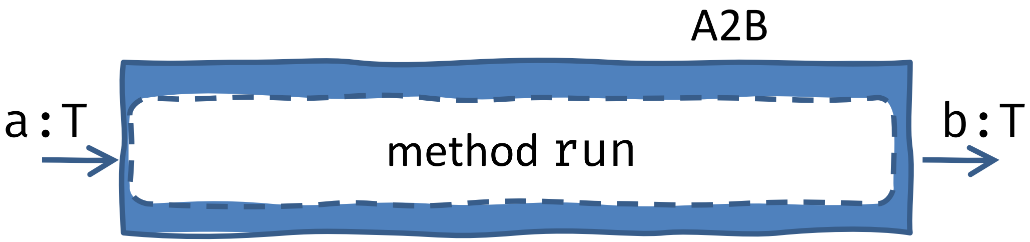
use clauses import cmt2’s eHDL and compiler features

itfc_declare! is a *macro*

macro in Rust transforms code at the token-level

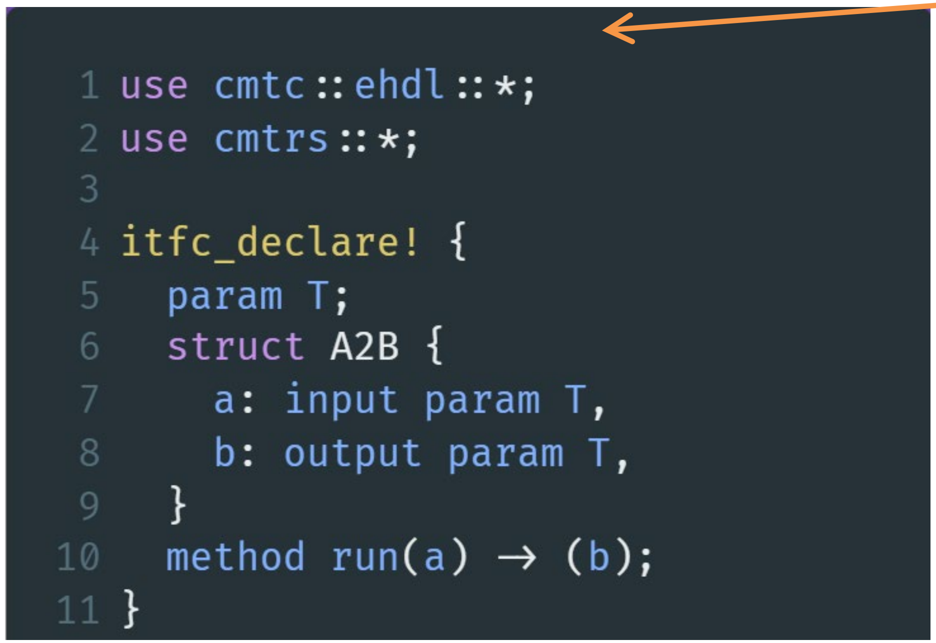
The left code creates …

- A module interface named A2B;
- It has one input port a, one output port b, both of which have type T.
- It exposes a method `run` for external world to invoke!

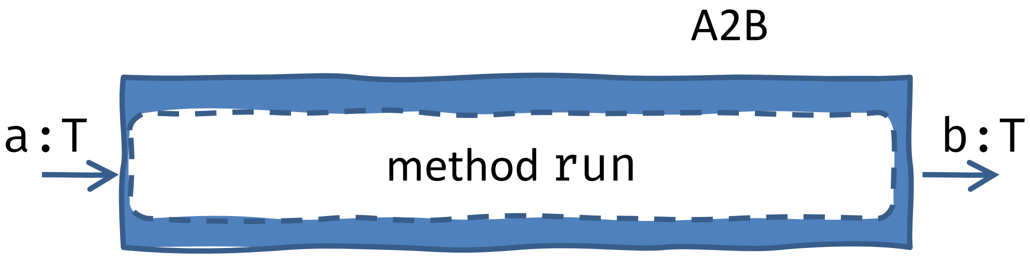


cmt2 is rule-based HDL

at the top of `hello.rs`



```rust
use cmtc::ehdl::*;
use cmtrs::*;

itfc_declare! {
  param T;
  struct A2B {
    a: input param T,
    b: output param T,
  }
  method run(a) → (b);
}
```



cmt2! *rule-based design*

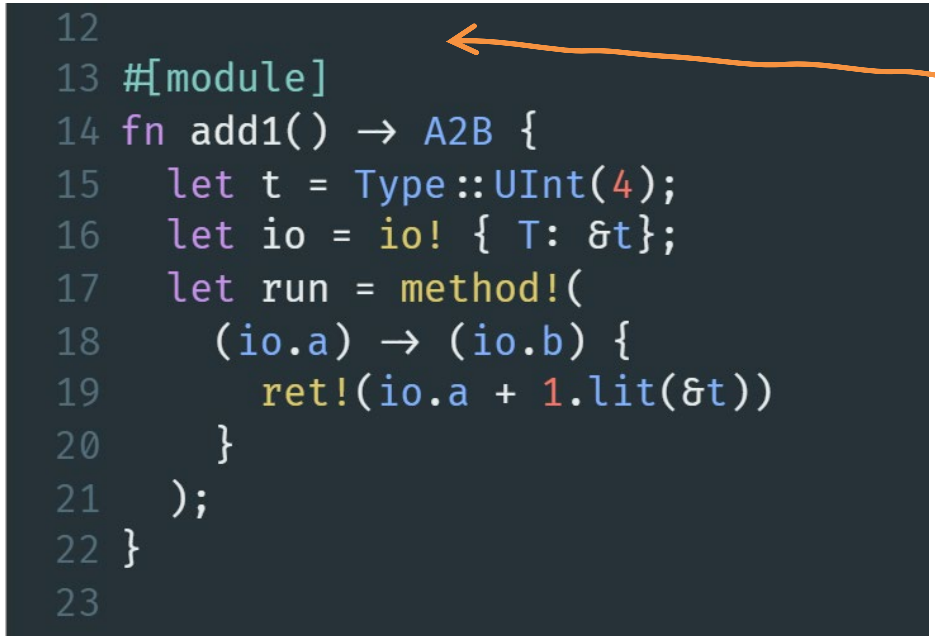
Hardware logic is organized as **rules** – the execution unit!

Don’t be afraid! 😁
Just consider one rule as a group of operations that execute **atomically**

In cmt2, we have two types of rules: *method* rules and *always* rules:

- *method* rule need to be invoked (Bluespec method)
- *always* rule runs actively (Bluespec rule)

cmt2 is rule-based HDL

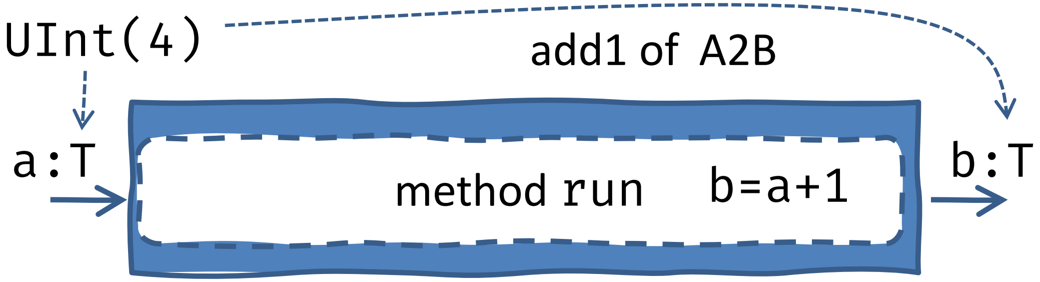


```
#[module]
fn add1() -> A2B {
  let t = Type::UInt(4);
  let io = io! { T: &t};
  let run = method!(
    (io.a) -> (io.b) {
      ret!(io.a + 1.lit(&t))
    }
  );
}
```

following `itfc_declare!`

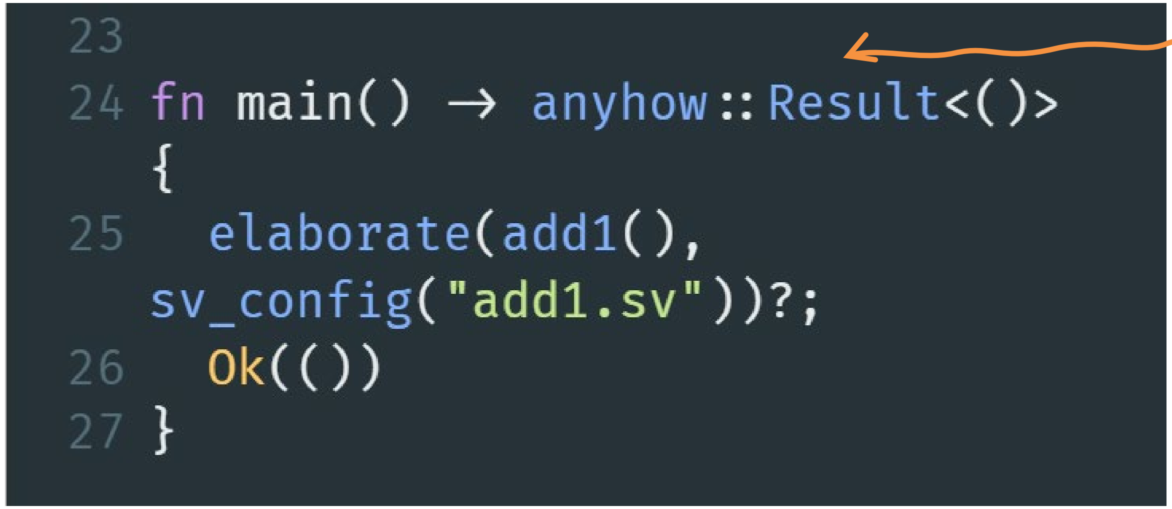
fn starts a function!

The `#[module]` *macro* transfoms the function `add1` to create a module of the interface A2B



Now, types are given, rules are filled -- Module is completely described!

cmt2 generates SystemVerilog



```rust

fn main() -> anyhow::Result<()> {
  elaborate(add1(), sv_config("add1.sv"))?;
  Ok(())
}
```

following `fn add1`

The new `main` function to call `elaborate` for the `add1` module

click [run] above `fn main`, or …

```
⌨ cargo run --example hello
⌨ cat add1.sv
```

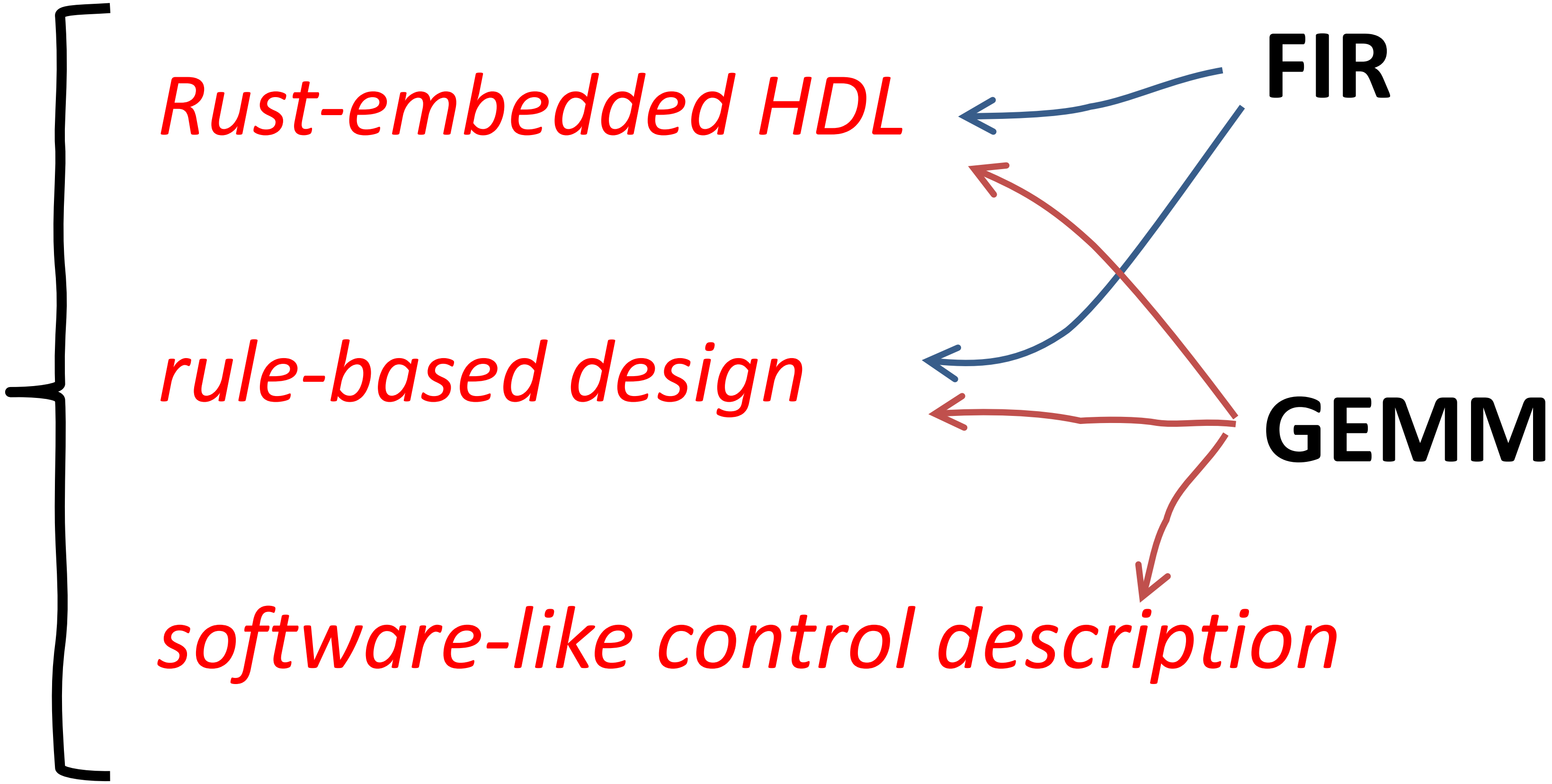
Show features by examples

as the successor of …

Cement (cmt1)

published at FPGA’24

cmt2!



Rust-embedded HDL

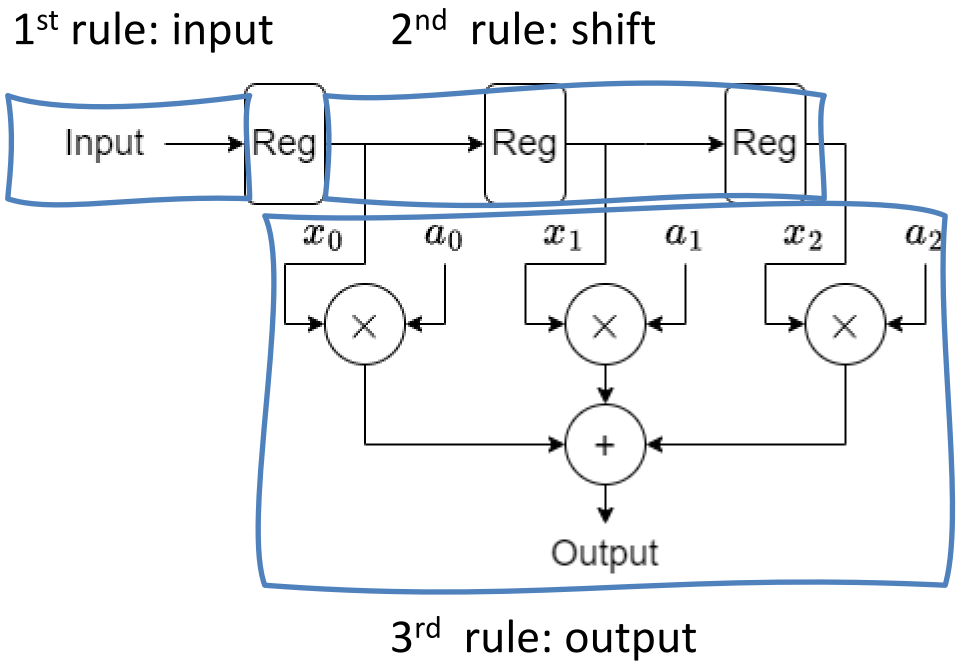
rule-based design

software-like control description

FIR

GEMM

Finite Impulse Response



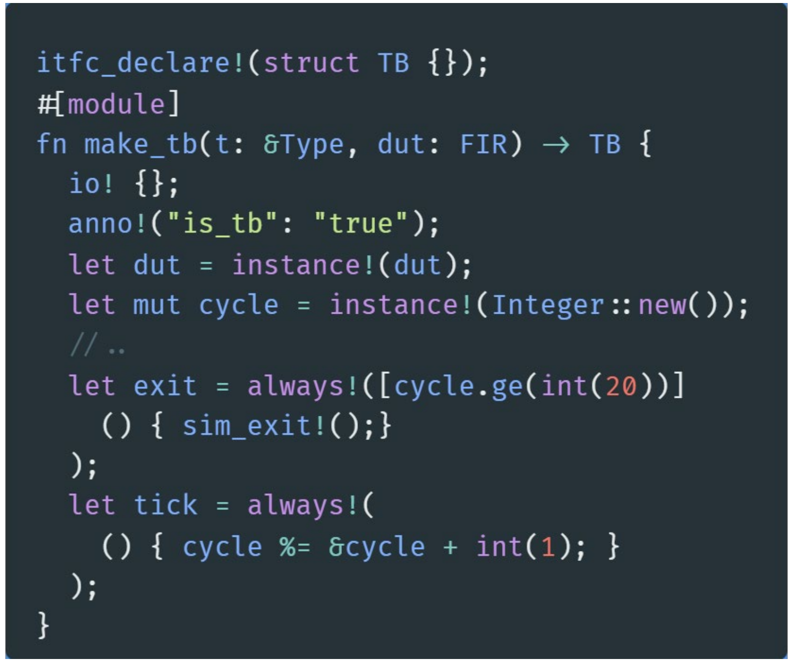
We provide three implementation

- `fir_3`: fixed length of 3
  ./crates/cmtc/examples/fir.rs line 18-46
- `gen_fir`: arbitrary length
  ./crates/cmtc/examples/fir.rs line 48-78
- `gen_fir_addertree`:
  arbitrary length with adder tree
  ./crates/cmtc/examples/fir.rs line 132-174

rule-based design

Rust-embedded HDL

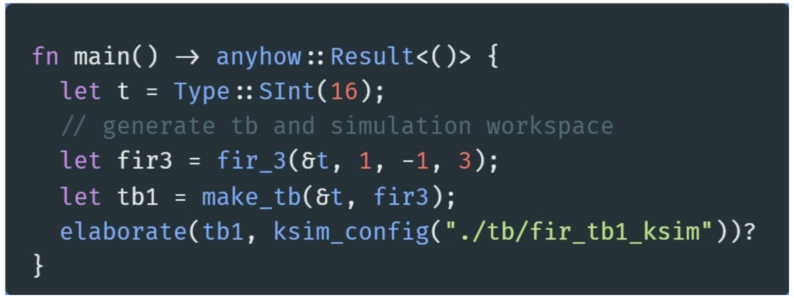
FIR Example: Testbench & Simulation



```
itfc_declare!(struct TB {});
#[module]
fn make_tb(t: &Type, dut: FIR) -> TB {
  io! {};
  anno!("is_tb": "true");
  let dut = instance!(dut);
  let mut cycle = instance!(Integer::new());
  // ..
  let exit = always!([cycle.ge(int(20))]
    () { sim_exit!();}
  );
  let tick = always!(
    () { cycle %= &cycle + int(1); }
  );
}
```

./crates/cmtc/examples/fir.rs   line 127-173

Tb is also described as **rules**, but will not be synthesized into hardware



```
fn main() -> anyhow::Result<()> {
  let t = Type::SInt(16);
  // generate tb and simulation workspace
  let fir3 = fir_3(&t, 1, -1, 3);
  let tb1 = make_tb(&t, fir3);
  elaborate(tb1, ksim_config("./tb/fir_tb1_ksim"))?
}
```

./crates/cmtc/examples/fir.rs   line 176-204

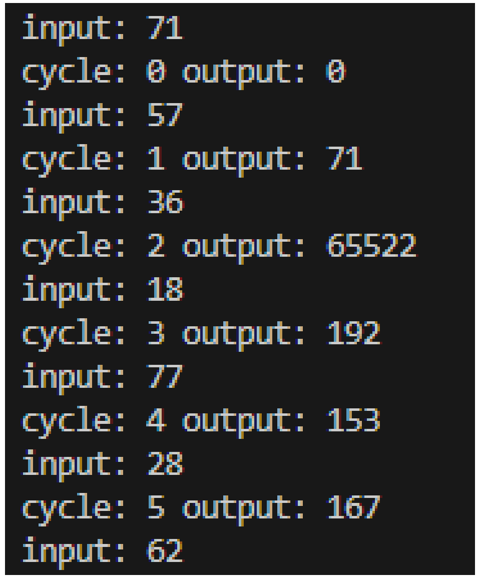
Elaboration creates a simulation directory

```
cd $REPOS/cmt2
cargo run --example fir
cd ./tb/fir_tb1_ksim
make all
./FIR_s6
```

run Rust

simulator make

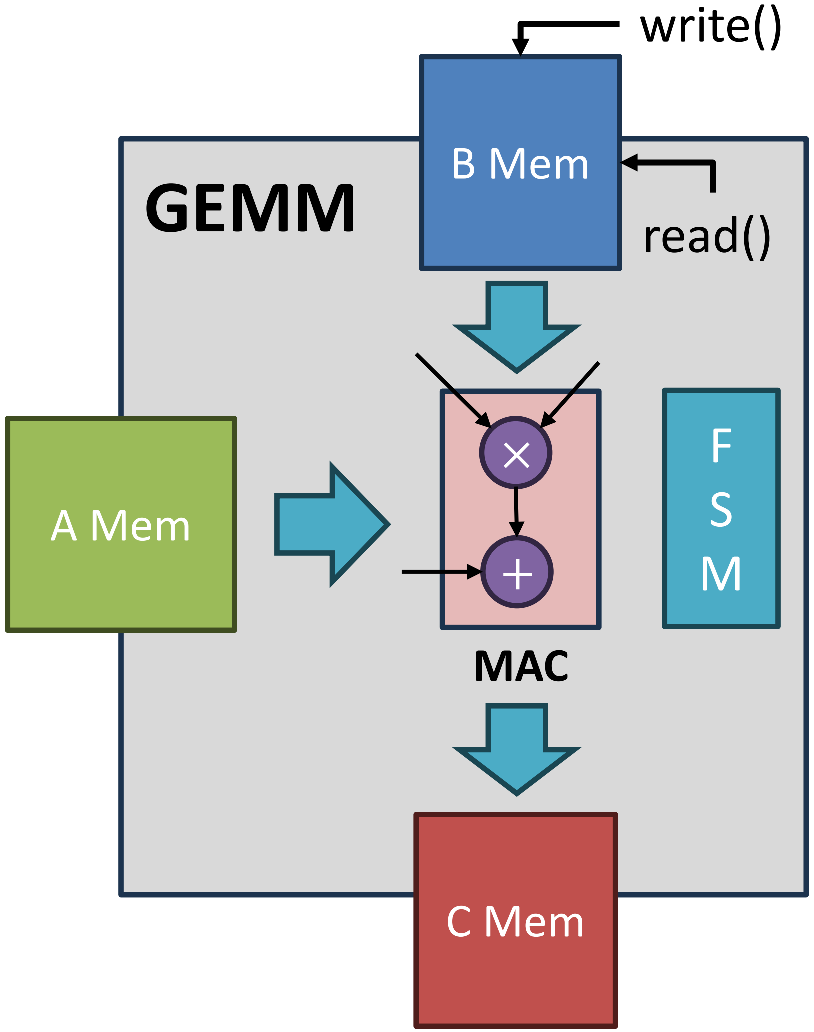
run the simulation program

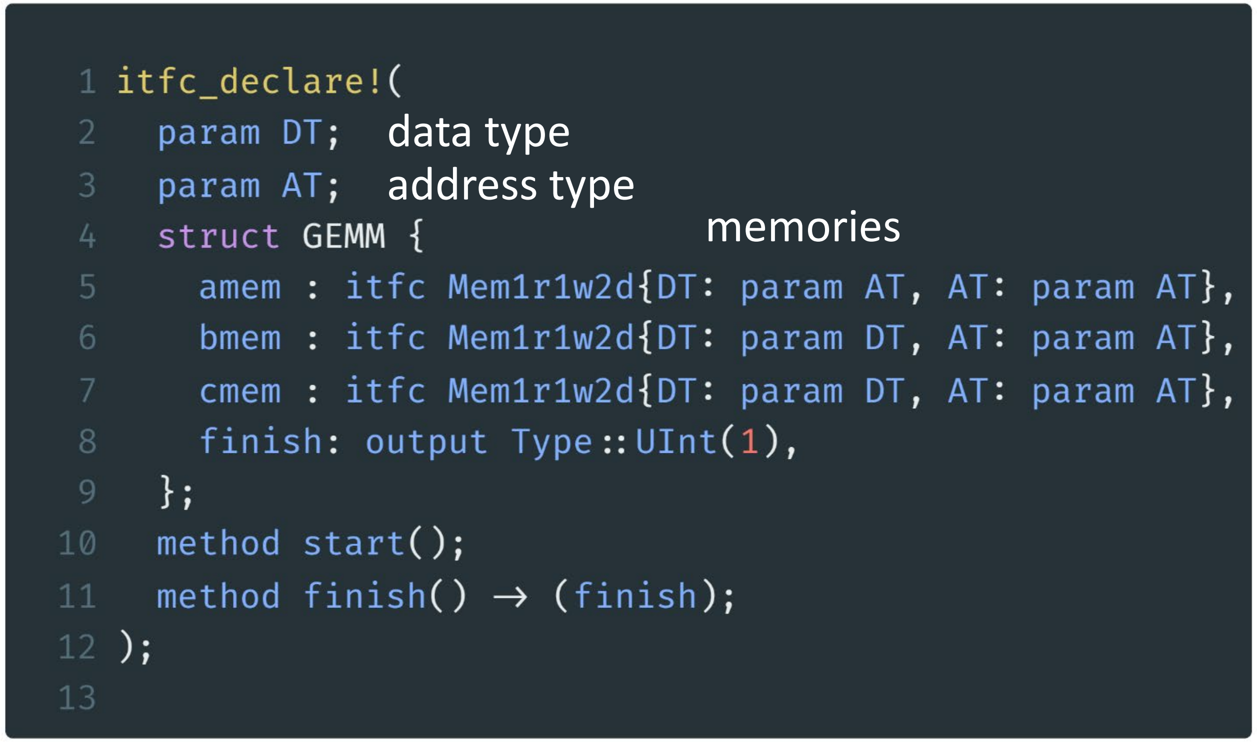


```
input: 71
cycle: 0 output: 0
input: 57
cycle: 1 output: 71
input: 36
cycle: 2 output: 65522
input: 18
cycle: 3 output: 192
input: 77
cycle: 4 output: 153
input: 28
cycle: 5 output: 167
input: 62
```

Possible Outputs

General Matrix Multiplication





```
itfc_declare!(
  param DT;
  param AT;
  struct GEMM {
    amem : itfc Mem1r1w2d{DT: param AT, AT: param AT},
    bmem : itfc Mem1r1w2d{DT: param DT, AT: param AT},
    cmem : itfc Mem1r1w2d{DT: param DT, AT: param AT},
    finish: output Type::UInt(1),
  };
  method start();
  method finish() -> (finish);
);
```

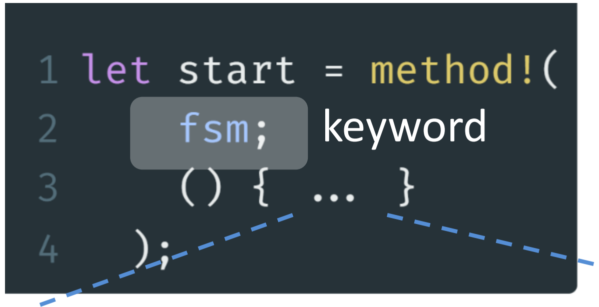
data type

address type

memories

./crates/cmtc/examples/gemm.rs
line 25-36

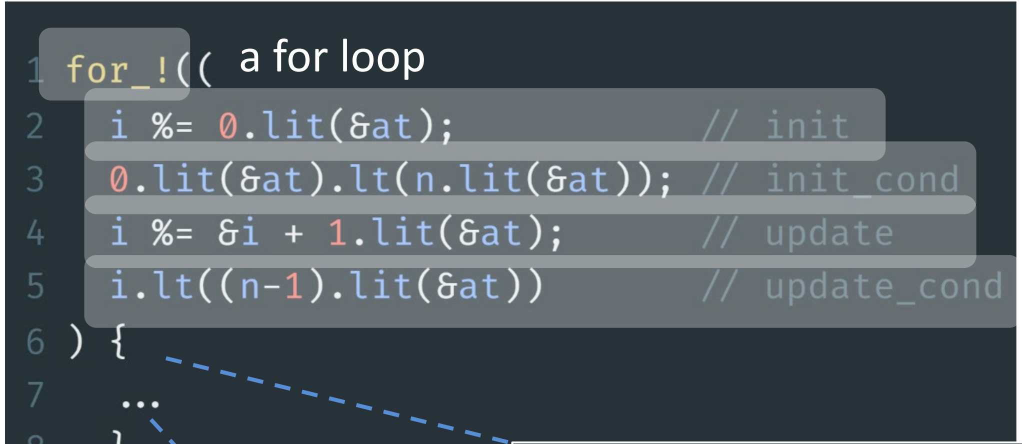
GeMM Example: FSM



```
let start = method!(
    fsm;
    () { ... }
  );
```

keyword

software-like control description



```
for_!((
  i %= 0.lit(&at);                    // init
  0.lit(&at).lt(n.lit(&at));          // init_cond
  i %= &i + 1.lit(&at);               // update
  i.lt((n-1).lit(&at))                // update_cond
) {
   ...
  }
```

a for loop

i= 0 to n-1

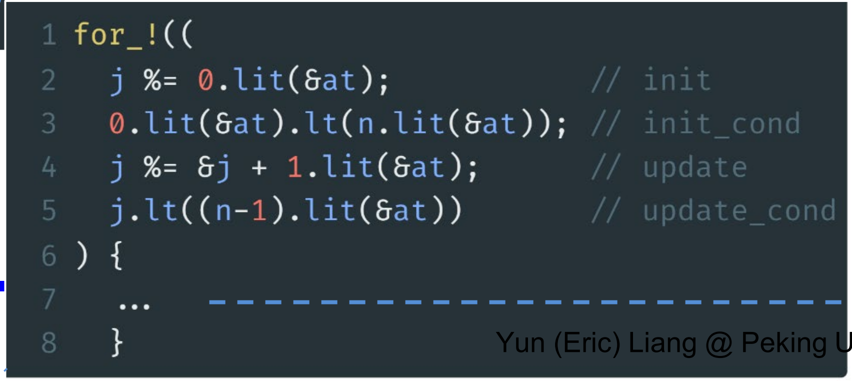
init: i=0

i<n *i=0* => 0<n

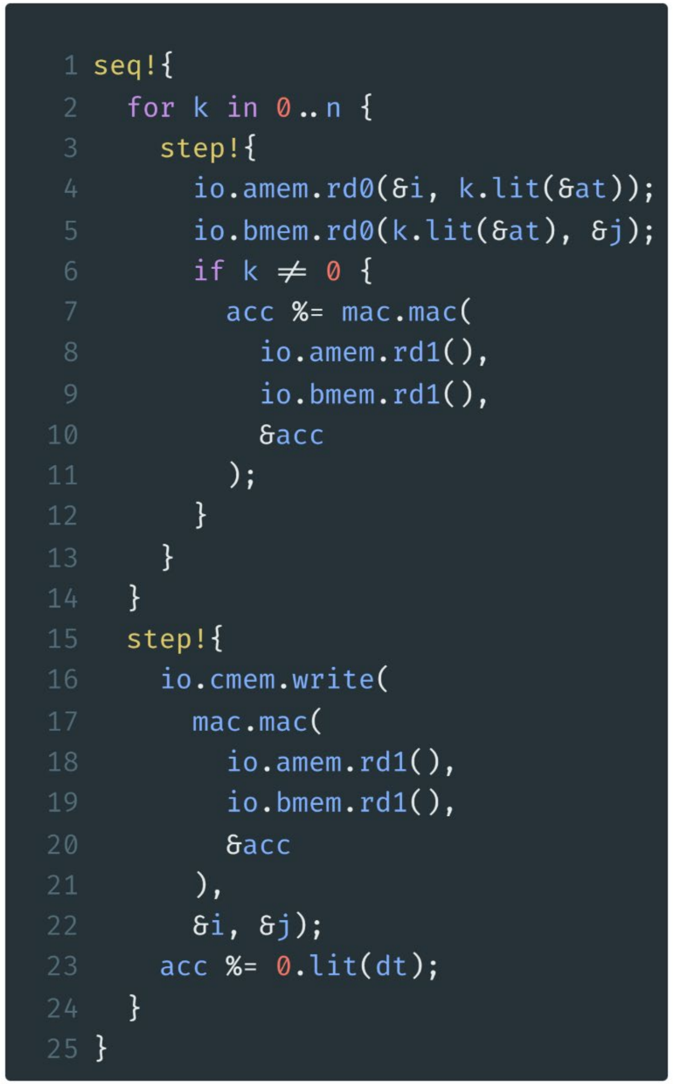
update: i=i+1

i<n *i=i+1* => i<n-1

j= 0 to n-1



```
for_!((
  j %= 0.lit(&at);                    // init
  0.lit(&at).lt(n.lit(&at));          // init_cond
  j %= &j + 1.lit(&at);               // update
  j.lt((n-1).lit(&at))                // update_cond
) {
   ...
  }
```

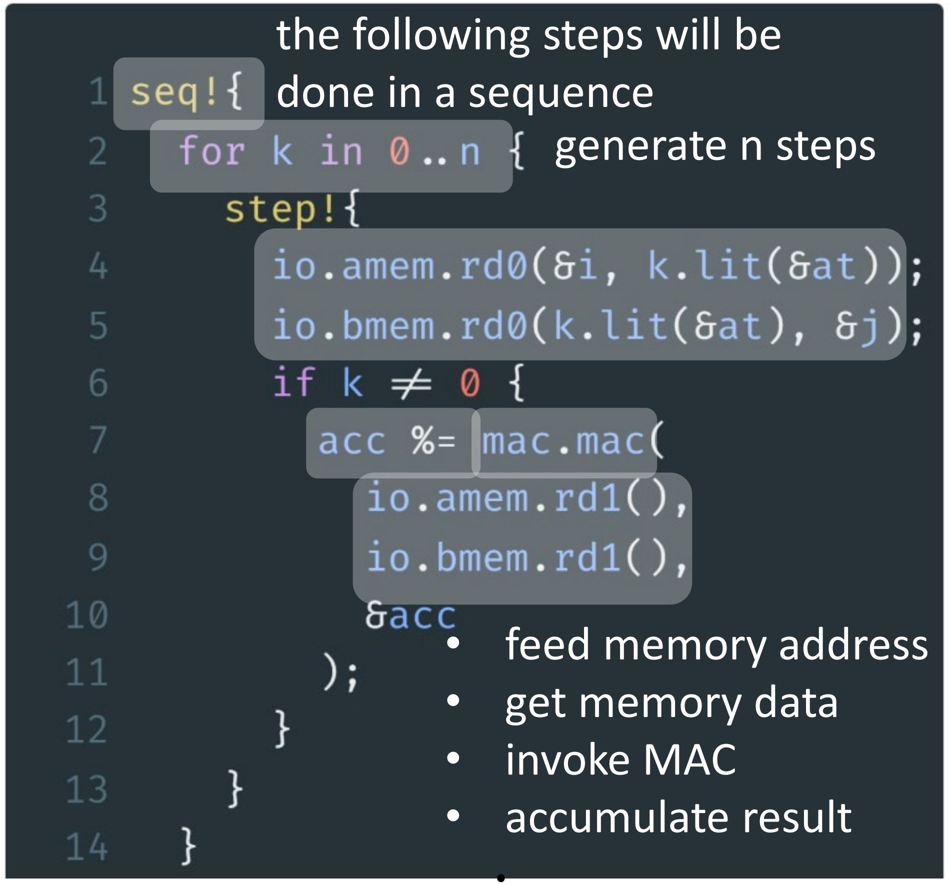


```
seq!{
  for k in 0..n {
    step!{
      io.amem.rd0(&i, k.lit(&at));
      io.bmem.rd0(k.lit(&at), &j);
      if k ≠ 0 {
        acc %= mac.mac(
          io.amem.rd1(),
          io.bmem.rd1(),
          &acc
        );
      }
    }
  }
  step!{
    io.cmem.write(
      mac.mac(
        io.amem.rd1(),
        io.bmem.rd1(),
        &acc
      ),
      &i, &j);
    acc %= 0.lit(dt);
  }
}
```

./crates/cmtc/examples/gemm.rs
line 38-97

GeMM Example: FSM innermost body

- for i in 0..n
  - for j in 0..n

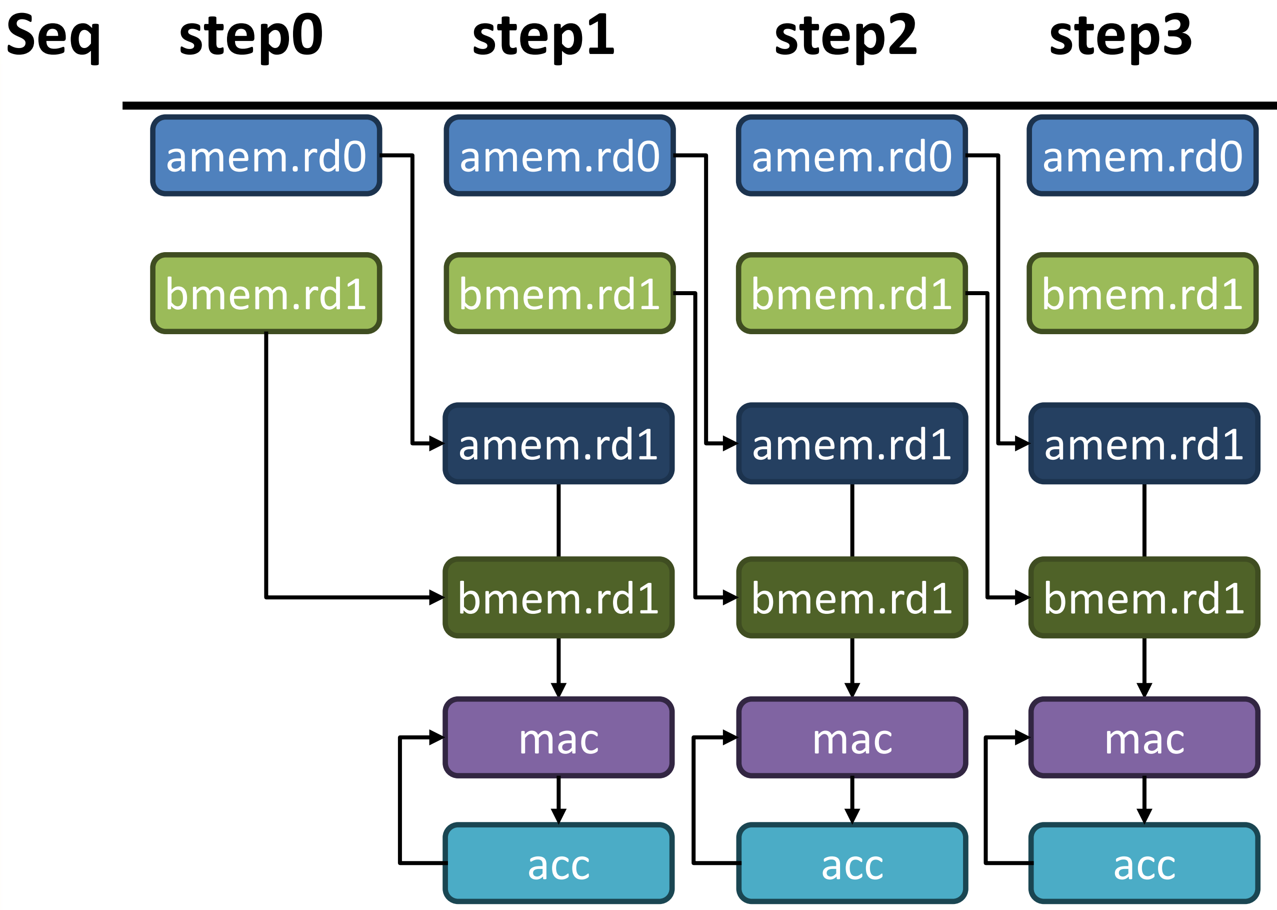


```
seq!{
  for k in 0..n {
    step!{
      io.amem.rd0(&i, k.lit(&at));
      io.bmem.rd0(k.lit(&at), &j);
      if k ≠ 0 {
        acc %= mac.mac(
          io.amem.rd1(),
          io.bmem.rd1(),
          &acc
        );
      }
    }
  }
```

the following steps will be done in a sequence

generate n steps

- feed memory address
- get memory data
- invoke MAC
- accumulate result



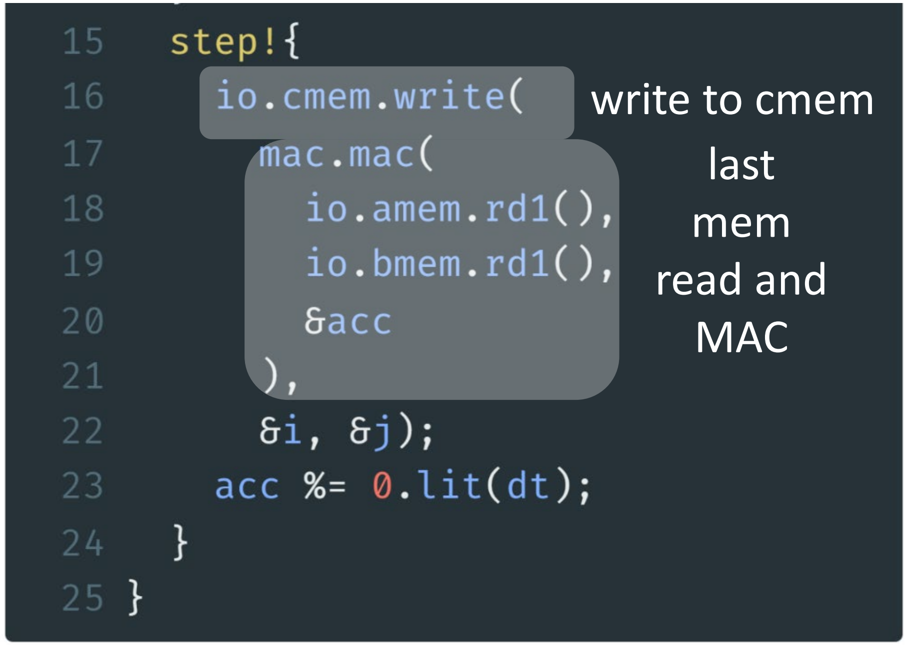
The same for all n steps

./crates/cmtc/examples/gemm.rs line 38-97

GeMM Example: FSM last step

- for i in 0..n
  - for j in 0..n

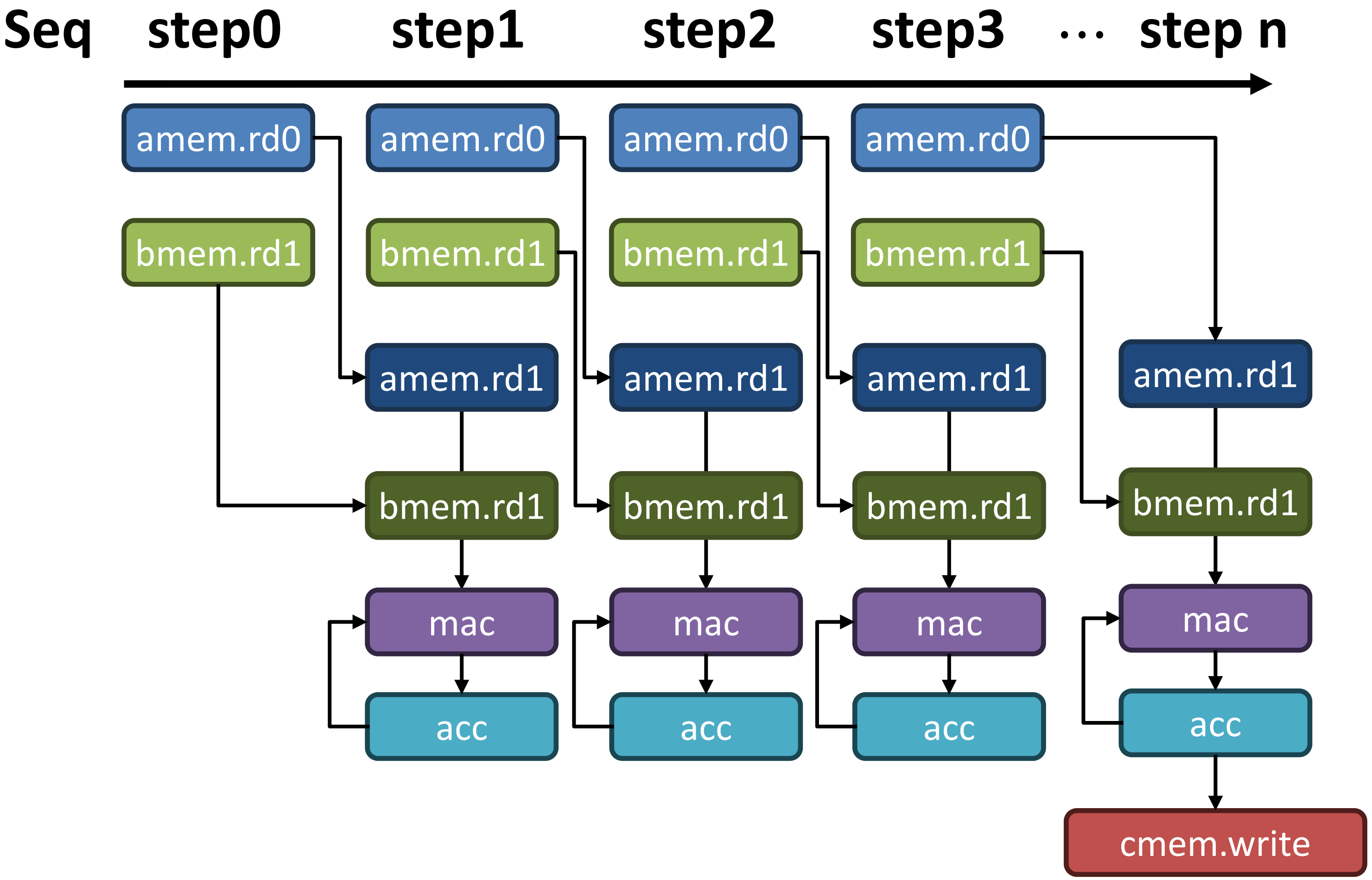
⋮



```
  step!{
    io.cmem.write(
      mac.mac(
        io.amem.rd1(),
        io.bmem.rd1(),
        &acc
      ),
      &i, &j);
    acc %= 0.lit(dt);
  }
}
```

write to cmem

last mem read and MAC

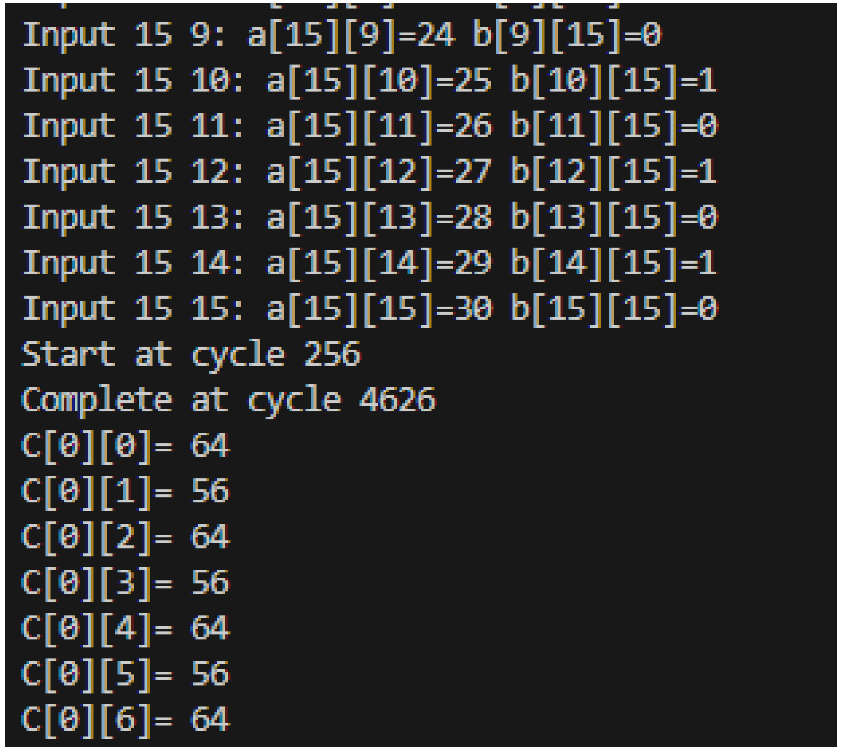


./crates/cmtc/examples/gemm.rs
line 38-97

GeMM Example: Simulation

⌨ `cd $REPOS/cmt2`
⌨ `cargo run --example gemm`
⌨ `cd ./tb/gemm_ksim`
⌨ `make all`
⌨ `./GEMM_s16_i8`

Commands to run simulation



```
Input 15 9: a[15][9]=24 b[9][15]=0
Input 15 10: a[15][10]=25 b[10][15]=1
Input 15 11: a[15][11]=26 b[11][15]=0
Input 15 12: a[15][12]=27 b[12][15]=1
Input 15 13: a[15][13]=28 b[13][15]=0
Input 15 14: a[15][14]=29 b[14][15]=1
Input 15 15: a[15][15]=30 b[15][15]=0
Start at cycle 256
Complete at cycle 4626
C[0][0]= 64
C[0][1]= 56
C[0][2]= 64
C[0][3]= 56
C[0][4]= 64
C[0][5]= 56
C[0][6]= 64
```

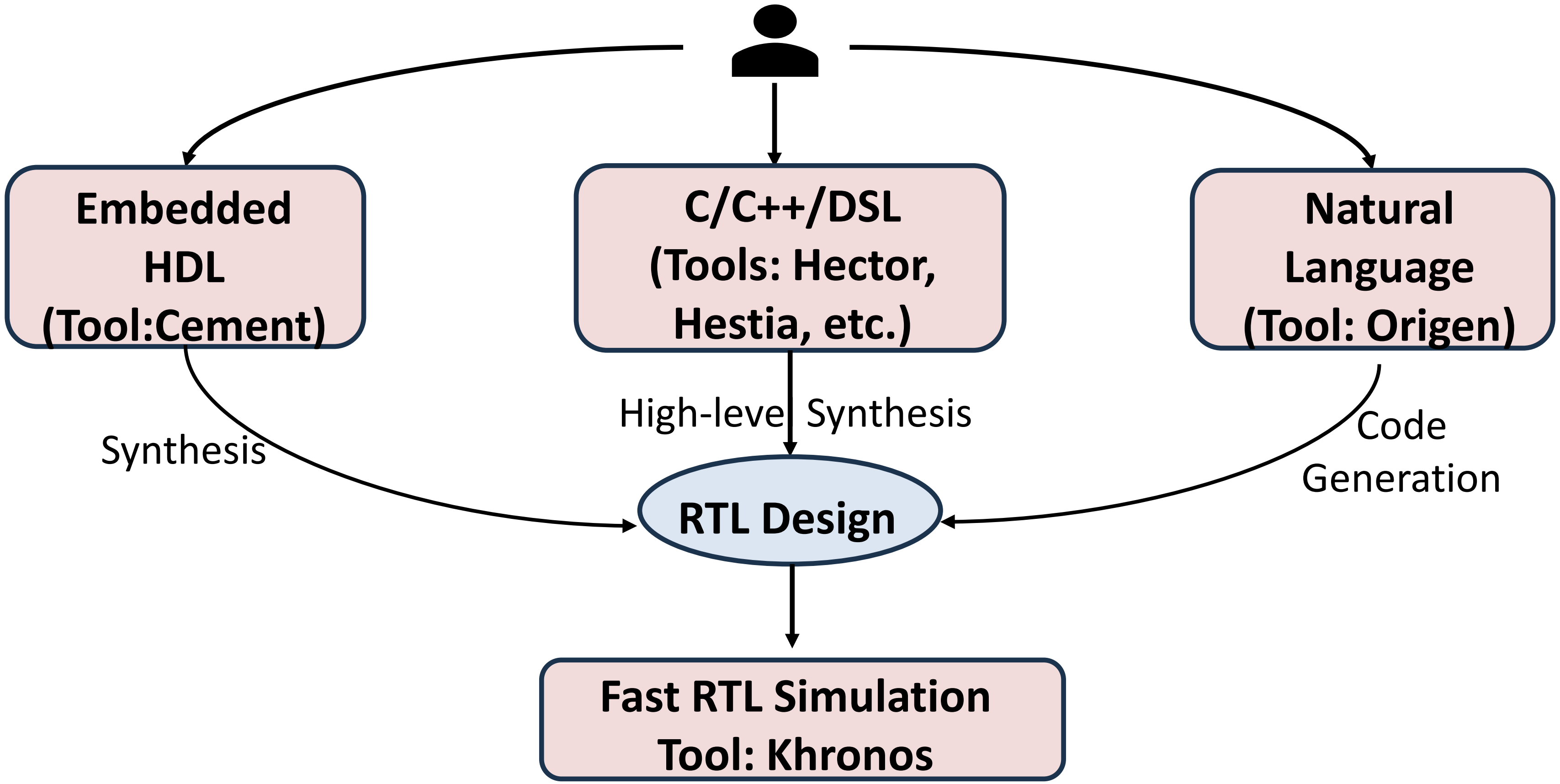
4370

Expected results

Cycles= $((16+1)$ //k
$\times 16+1)$ //j + 1 cycle to init
$\times 16+1$ //i + 1 cycle to init
= 4369 + 1 cycle to write finish

Summary

- Webpage: https://ericlyun.me/tutorial-date2025/
  - Papers, presentation, code



Embedded HDL (Tool:Cement)
C/C++/DSL (Tools: Hector, Hestia, etc.)
Natural Language (Tool: Origen)
Synthesis
High-level Synthesis
Code Generation
RTL Design
Fast RTL Simulation Tool: Khronos

Simulation Speedup

```
cd hestia/tutorial/benchmarks/collection1

time hestia command/aeloss_push/tor.tcl

time hestia command/aeloss_push/hec.tcl
```

	TOR IR	HEC IR
Cycle	1502609	1502706
Time	2.019s	37.711s