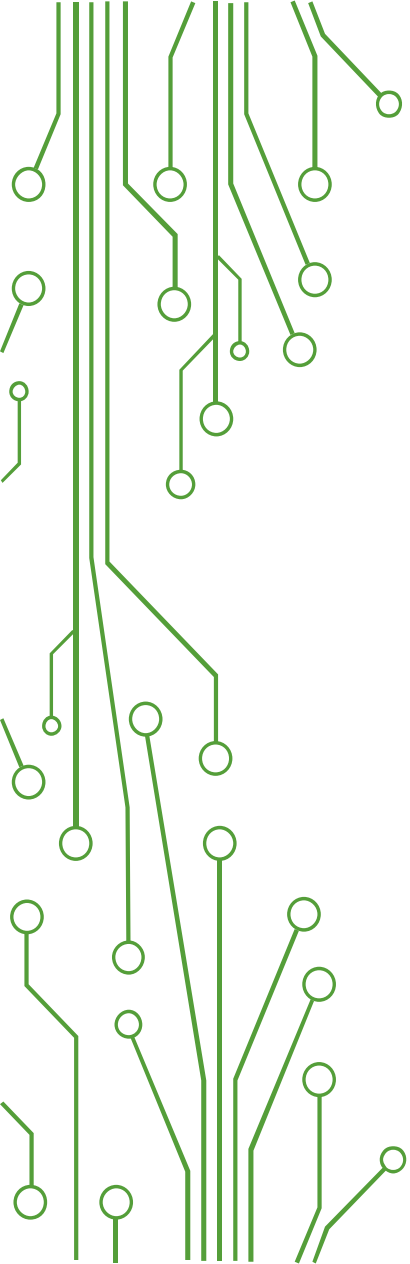


北京大学高能效计算与应用中心
Center for Energy-efficient Computing and Applications

高层综合与AIE阵列映射的MLIR应用

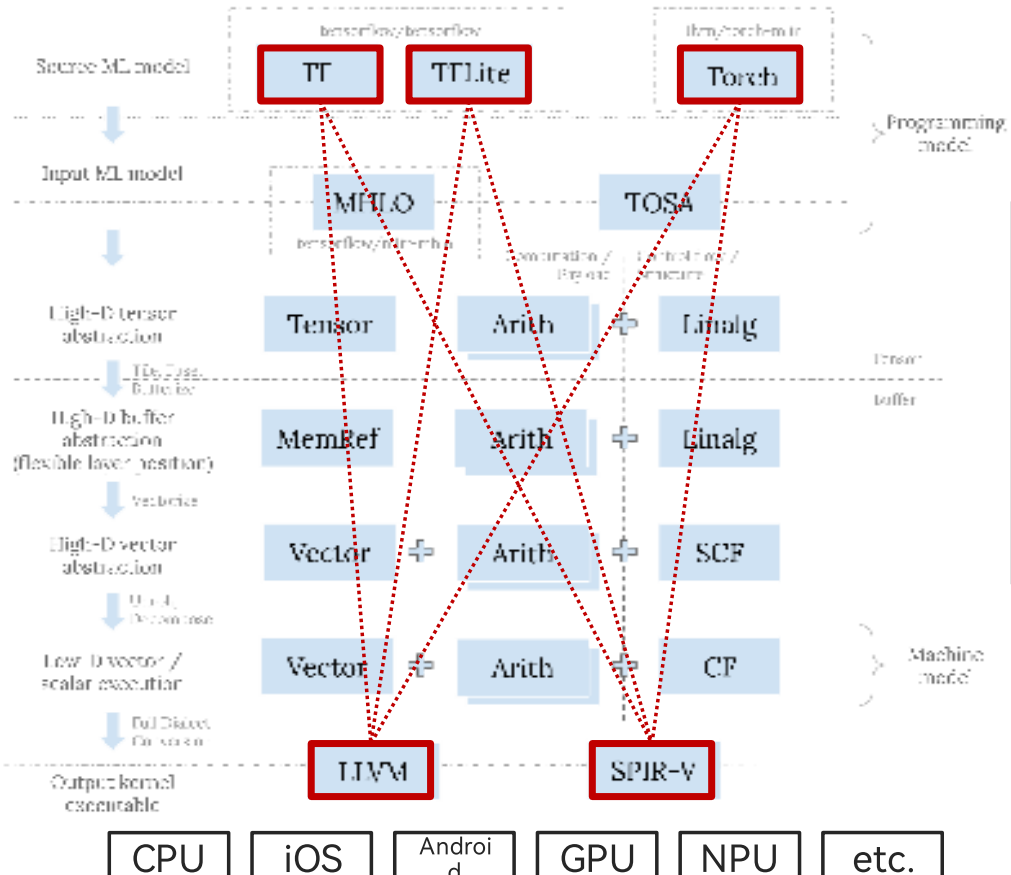
罗国杰

gluo@pku.edu.cn

北京大学

Why Compiler Infrastructures? “p×n×q” → “p+n+q”

image source: lei.chat



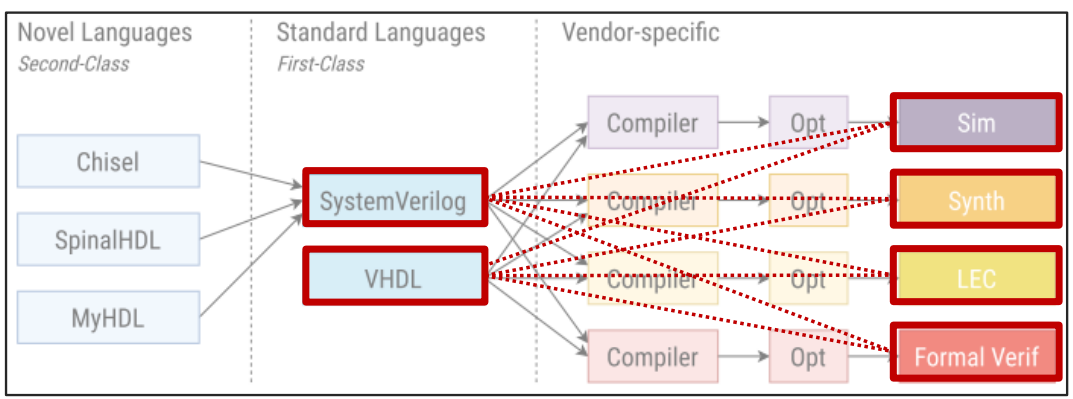


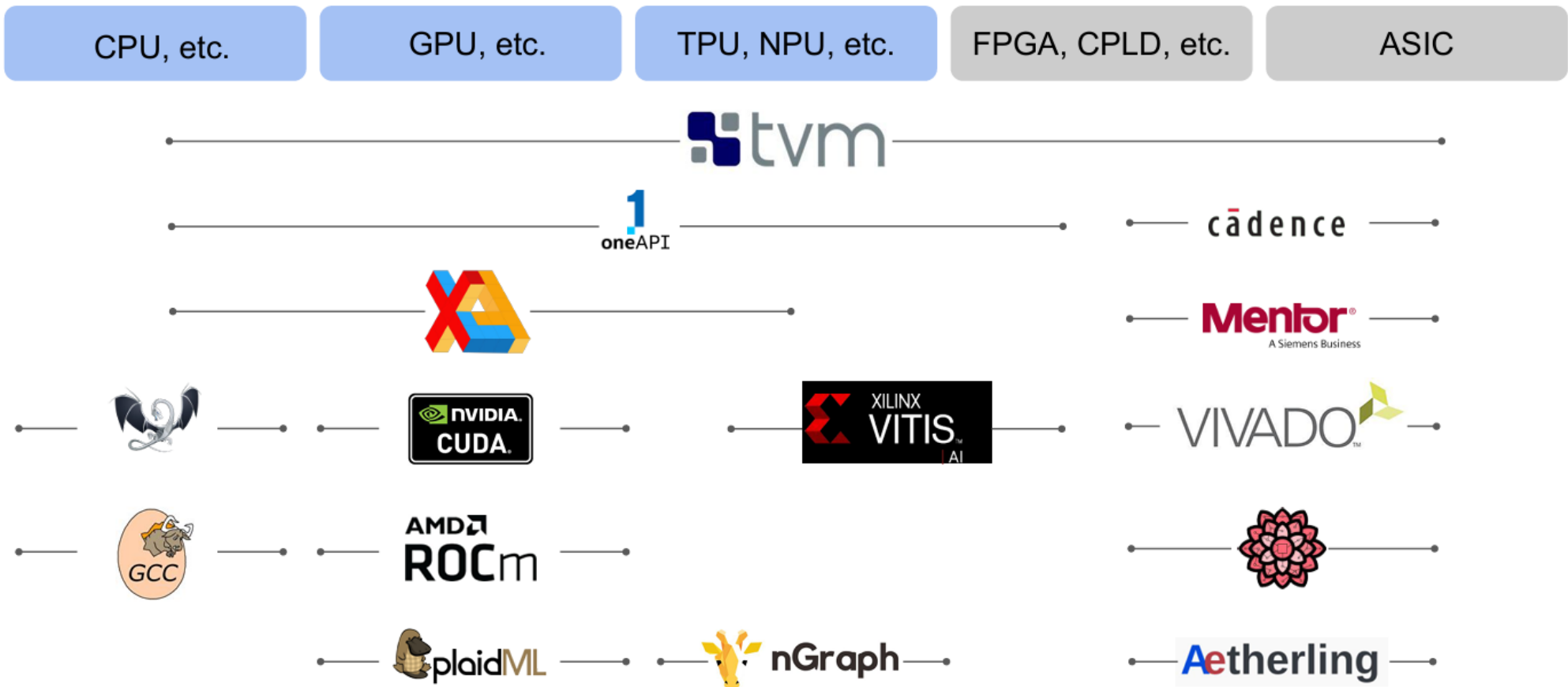
image source: [LLHD@PLDI’20]

Refactor: $x_1y_1z_1 + x_1y_1z_2 + x_1y_2z_1 + x_1y_2z_2 + x_1y_3z_1 + x_1y_3z_2 + x_2y_1z_1 + x_2y_1z_2 + x_2y_2z_1 + x_2y_2z_2 + x_2y_3z_1 + x_2y_3z_2 = (x_1 + x_2) \cdot (y_1 + y_2 + y_3) \cdot (z_1 + z_2)$

How MLIR and CIRCT Fit In

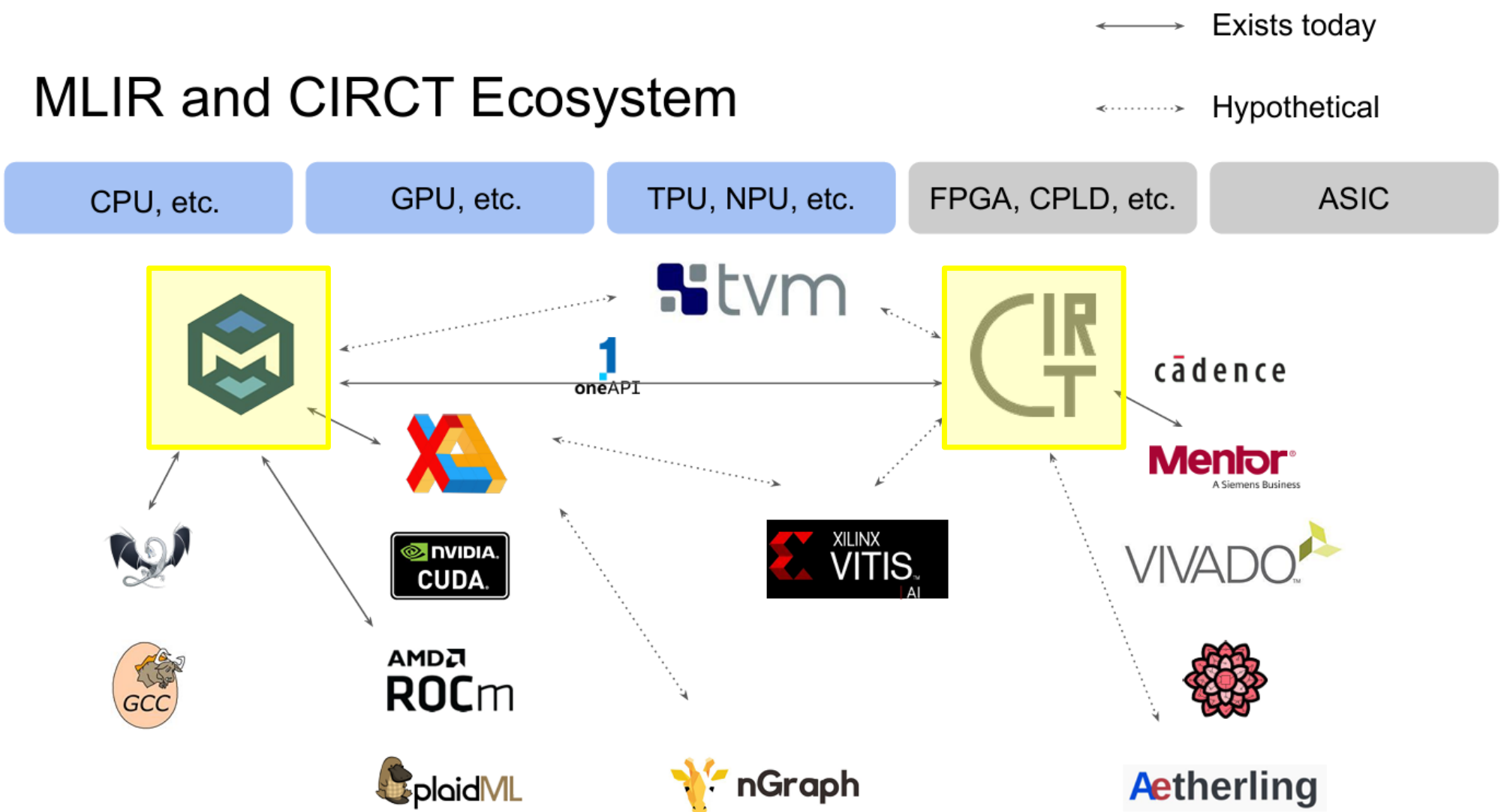
Fixed processor

Custom processor



Mike Urbach, “Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications,” MLSys Symposium: Chips and Compilers, 2021.

MLIR and CIRCT Ecosystem



Mike Urbach, “Applying Circuit IR Compilers and Tools (CIRCT) to ML Applications,” MLSys Symposium: Chips and Compilers, 2021.

What is MLIR for?

A hybrid IR to support multiple different requirements in a unified infrastructure:

- The ability to represent dataflow graphs (such as in TensorFlow)
- Optimizations and transformations typically done on such graphs (e.g. in Grappler).
- Ability to host high-performance loop optimizations and to transform memory layouts of data.
- Code generation “lowering” transformations (e.g., DMA insertaion, memory tiling, vectorization).
- Ability to represent target-specific operations, e.g. accelerator-specific high-level operations.
- Quantization and other graph transformations done on a Deep-Learning graph.
- Polyhedral primitives.
- **Hardware Synthesis Tools / HLS.**

Non-goals

- not for low level machine code generation algorithms (like reg. allocation and instr. scheduling).
- not a source language to write kernels (analogous to CUDA C++).

* https://mlir.llvm.org/
* C. Lattner et al., “MLIR: Scaling Compiler Infrastructure for Domain Specific Computation,” CGO 2021.

Users of MLIR

CIRCT: Circuit IR Compilers and Tools

- https://circt.llvm.org/
- The CIRCT project is an (experimental!) effort looking to apply MLIR and the LLVM development methodology to the domain of hardware design tools.

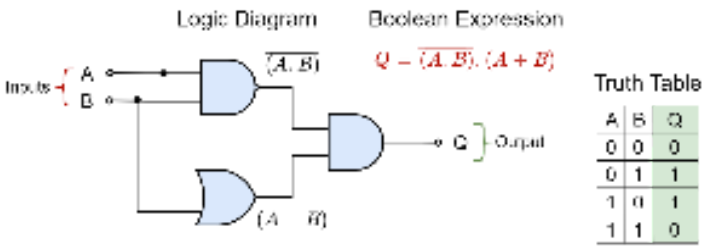
MLIR-AIE: Toolchain for AMD AI Engine devices

- https://xilinx.github.io/mlir-aie/
- MLIR-AIE is a toolchain providing low-level device configuration for Versal AIEngine-based devices. Support is provided to target the AIEngine portion of the device, including processors, stream switches, TileDMA and ShimDMA blocks. Backend code generation is included, targetting the LibXAIE library, along with some higher-level abstractions enabling higher-level design.

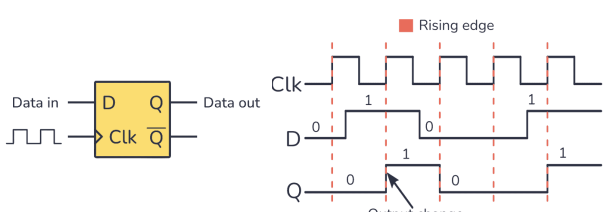
Background: Functionality of Digital Circuits

- Combinational (stateless) circuits
  - with input x and output y

$$y = f(x)$$

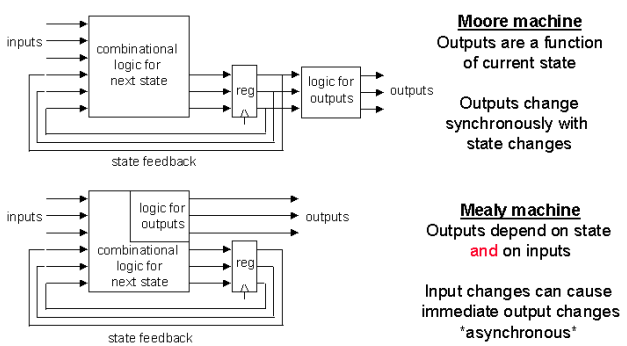


- Sequential (stateful) circuits
  - with input x, state S, and output y



$$\begin{cases} y = f(S) \\ S' = g(S,x) \end{cases} \quad \text{(Moore machine)}$$

$$\begin{cases} y = f(S,x) \\ S' = g(S,x) \end{cases} \quad \text{(Mealy machine)}$$



- Formally, $f(\cdot)$ and $g(\cdot)$ are 4-valued {0,1,X,Z} logic, instead of Boolean, functions

What is missing? Timing, Power, Languages (SystemVerilog/VHDL), etc.

Background: High-Level Synthesis (HLS) and HLS C

HLS C

- A hardware synthesis language with automatic scheduling, array partitioning, interface generation, etc., in C syntax

HLS vs. compilation

- Compilation: C ⇒ instruction stream
  - To compute, a CISC/RISC processor executes the instructions
- HLS: C ⇒ hardware description language (HDL)
  - To compute, the HDL-described circuit executes Boolean switching

Background: HLS C (in AMD/Xiinx Vitis as example)

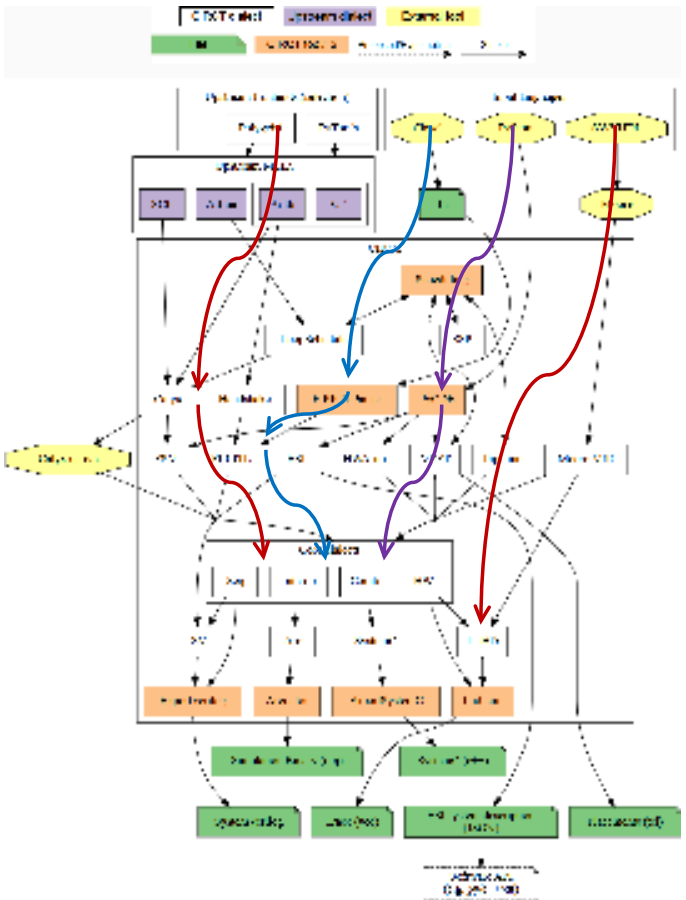
HLS C is a synthesizable and functional hardware description language

(functional: 功能性 not 函数式)

- Part I: C as the host language
  - functions ↔ modules; arguments ↔ I/O ports; control flows ↔ control logic;
  - operators ↔ functional units; scalar variables ↔ wires or regs; array variables ↔ on-chip memories;
- Part II: HLS pragmas https://docs.amd.com/r/en-US/ug1399-vitis-hls/HLS-Pragmas
  - interface synthesis: interface, stream
  - schedule-related: pipeline, dataflow
  - resource-related: unroll, array_partiton, allocation
- Part III: HLS library
  - HLS Arbitrary Precision Types Library https://github.com/Xilinx/HLS_arbitrary_Precision_Types/
  - HLS Stream Library https://github.com/Xilinx/hls-lib-stream/

  (The two repositories above is for simulation only. The synthesis is likely hardcoded in Vitis HLS tools.)

Circuit IR Compilers and Tools (CIRCT)



High-level IR

- Calyx
- Scheduling
- SSP
- LoopSchedule
- Pipeline
- Handshake
- ESI

RTL-synth IR

- FSM
- FIRRTL
- HWArith
- Core dialects
- SV
- MSFTs

RTL-sim IR

- Arc
- SystemC
- LLHD

https://circt.llvm.org/

High-Level Synthesis (HLS) in CIRCT

doHLSFlowDynamic()

- Software lowering
  - lower affine
  - Convert SCF to CF
  - DHLS pipeline
  - handshake tranforms pipeline
- HW path
  - simple canonicalizer
  - handshake to HW
  - simple canonicalizer
  - ESI lowering pipeline
  - HW lowering pipeline
  - export Verilog

doHLSFlowCalyx()

- Software transformations
  - SCF to Calyx; simple canonicalizer
  - remove comb. groups; simple canonicalizer
  - Calyx to FSM; simple canonicalizer
  - materialize Calyx to FSM; simple canonicalizer
  - remove groups from FSM; simple canonicalizer
- HW path
  - Calyx to HW; simple canonicalizer
  - convert FSM to SV
  - HW lowering pipeline
  - export Verilog

https://github.com/llvm/circt/blob/main/tools/hlstool/hlstool.cpp

HLS in CIRCT

```
cgeist -S --function=* --memref-fullrank --scal-rep ${design}.c | \
mlir-opt --scf-for-to-while | \
circt-opt --lower-scf-to-calyx --canonicalize | \
circt-translate --export-calyx | \
calyx -b verilog
```

[lower-scf-to-fsm] error: Operation 'calyx.par' not supported for FSM compilation #5154
https://github.com/llvm/circt/issues/5154

[CalyxToFSM] Add par FSM lowering #4606
https://github.com/llvm/circt/issues/4606

doHLSFlowCalyx()

- Software transformations
  - SCF to Calyx; simple canonicalizer
  - remove comb. groups; simple canonicalizer
  - Calyx to FSM; simple canonicalizer
  - materialize Calyx to FSM; simple canonicalizer
  - remove groups from FSM; simple canonicalizer
- HW path
  - Calyx to HW; simple canonicalizer
  - convert FSM to SV
  - HW lowering pipeline
  - export Verilog

HLS in CIRCT

doHLSFlowDynamic()

- Software lowering
  - lower affine
  - Convert SCF to CF
  - DHLS pipeline
  - handshake tranforms pipeline
- HW path
  - simple canonicalizer
  - handshake to HW
  - simple canonicalizer
  - ESI lowering pipeline
  - HW lowering pipeline
  - export Verilog

```
cgeist -S --function=* --memref-fullrank -scal-rep ${design}.c | \
mlir-opt --lower-affine | \
mlir-opt --convert-scf-to-cf | \
circt-opt --flatten-memref | \
circt-opt --handshake-legalize-memrefs | \
mlir-opt --convert-scf-to-cf --canonicalize | \
circt-opt --lower-cf-to-handshake | \
circt-opt --handshake-lower-extmem-to-hw --canonicalize | \
circt-opt --handshake-materialize-forks-sinks --canonicalize | \
circt-opt --handshake-insert-buffers --canonicalize | \
circt-opt --lower-handshake-to-hw --canonicalize | \
circt-opt --lower-esi-ports | \
circt-opt --lower-esi-to-physical | \
circt-opt --lower-esi-to-hw --canonicalize | \
circt-opt --lower-seq-hlmem | \
circt-opt --hw-memory-sim | \
circt-opt --lower-seq-to-sv | \
circt-opt --hw-cleanup | \
circt-opt --hw-legalize-modules --canonicalize | \
circt-opt --prettify-verilog | \
circt-opt --export-verilog
```

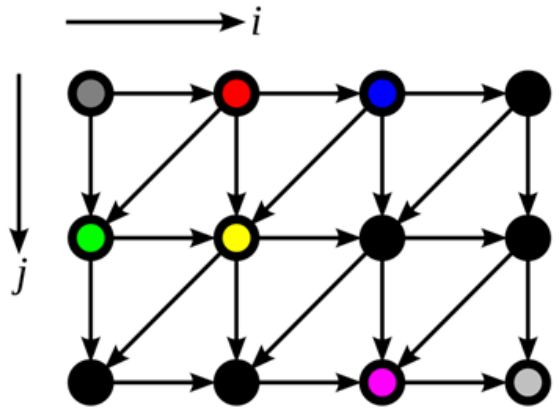
MLIR: Affine Dialect

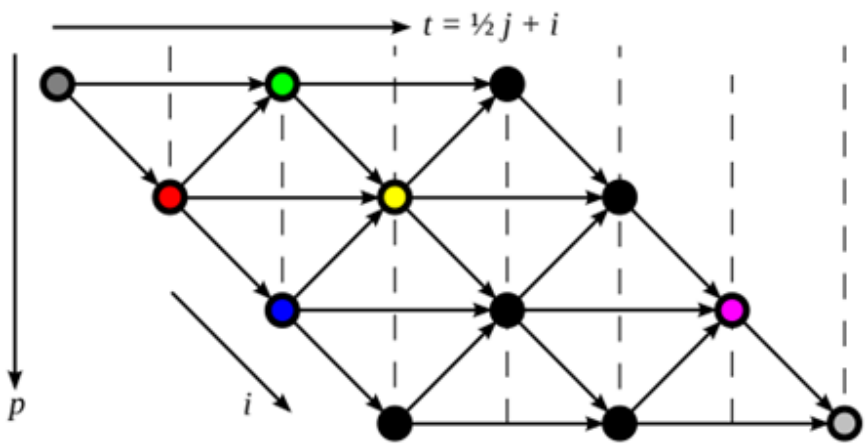
Polyhedral compilation toolbox

Affine expressions, maps, and integer sets

Affine loops and transformations

```
func @matmul(%A: memref<5x10xf32>, %B: memref<10x10xf32>,
             %C: memref<10x10xf32>) -> memref<10x10xf32> {
  affine.for %arg3 = 0 to 5 {
    affine.for %arg4 = 0 to 10 {
      affine.for %arg5 = 0 to 10 { ... }
    }
  }
  return %0 : memref<10x10xf32>
}
```





MLIR: SCF and CF Dialects

Structured Control Flow (SCF) :

- Contains operations that represent control flow constructs such as if and for
- The control flow has a structure unlike, for example, gotos or asserts

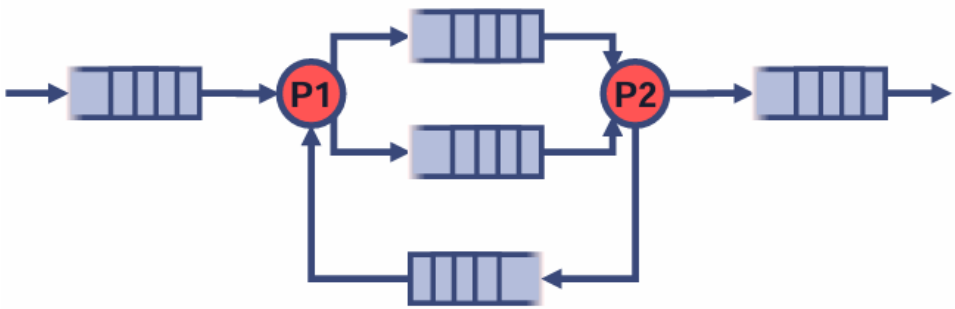
```
scf.for %iv = %lb to %ub step %step {
  ... // body
}
```

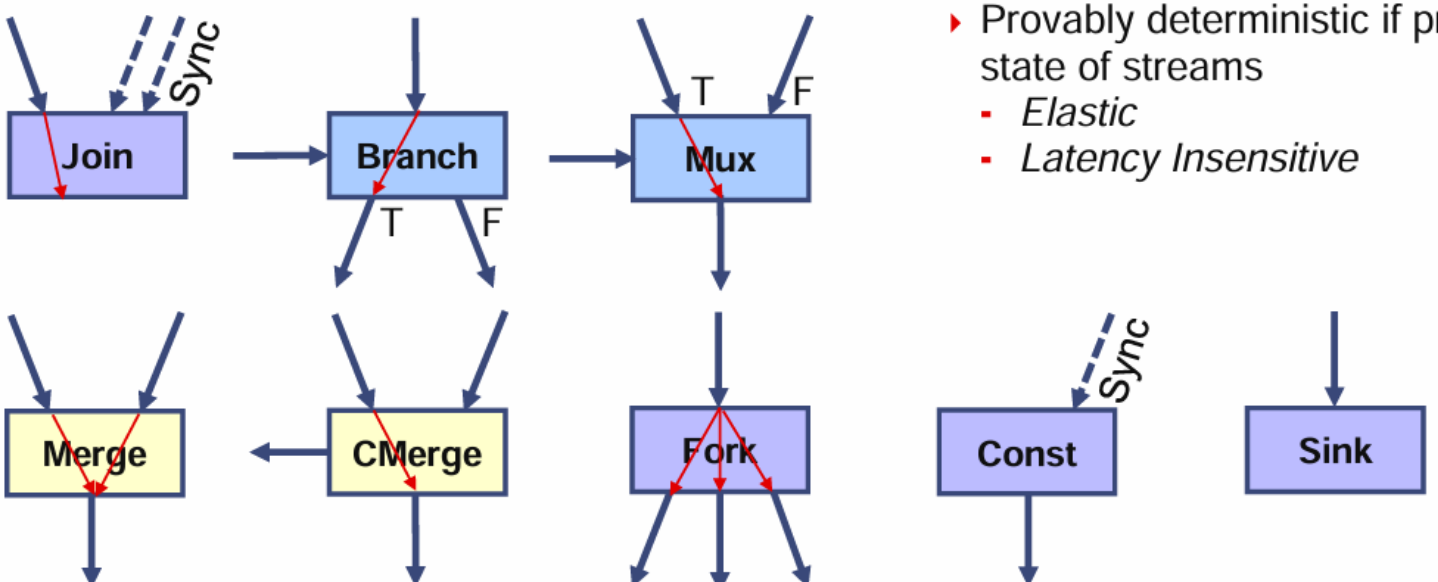
Control Flow (CF):

- Contains low-level, i.e. non-region based, control flow constructs
- Represents control flow directly on SSA blocks

```
^bb2:
  %2 = call @someFn()
  cf.br ^bb3(%2 : tensor<*xf32>)
^bb3(%3: tensor<*xf32>):
```

CIRCT: Handshake Dialect



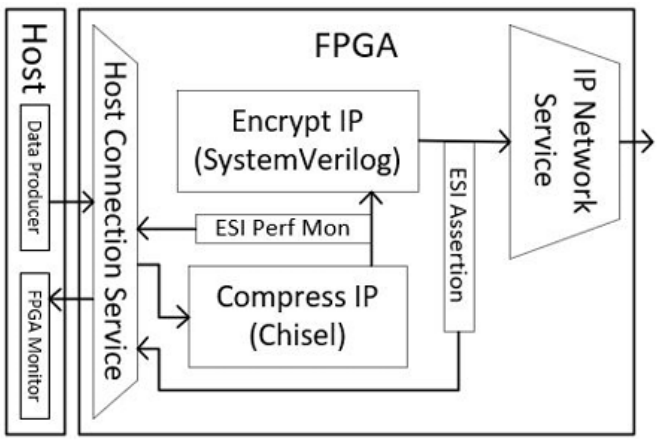


- Processes communicate through stream interfaces (e.g. AXI4-Stream)
- Interfaces connected by single-reader single-writer FIFOs, which are logically unbounded
- Processes can access interfaces in any order
- Provably deterministic if processes cannot test state of streams
  - *Elastic*
  - *Latency Insensitive*

Hanchen Ye, "Handshake-based HLS in CIRCT," Circuit IR Compilers and Tools Open Meeting, 2020.

CIRCT: ESI Dialect

- Type system for channels in hardware
- Connect on- and off-chip components
- Generates elastic connector circuits
- Supports cosimulation




```
rtl.module @esi(%clock: i1, %reset: i1) {
  %esiChannel, %0 = rtl.instance "sender" @Sender (%clock) : (i1) -> (!esi.channel<i4>, i8)
  %bufferedChannel = esi.buffer %clock, %reset, %esiChannel { stages = 4 } : i4
  rtl.instance "receiver" @Reciever (%bufferedChannel, %clock) : (!esi.channel<i4>, i1) -> ()
}
```

https://circt.llvm.org/docs/Dialects/ESI/RationaleESI/

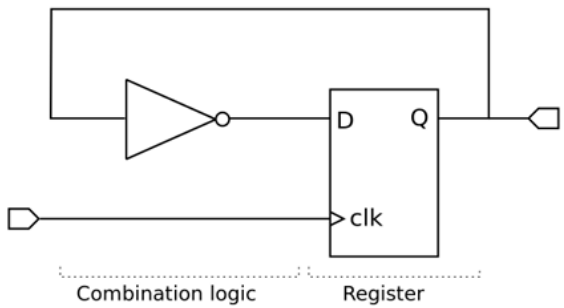
CIRCT: Core Dialects

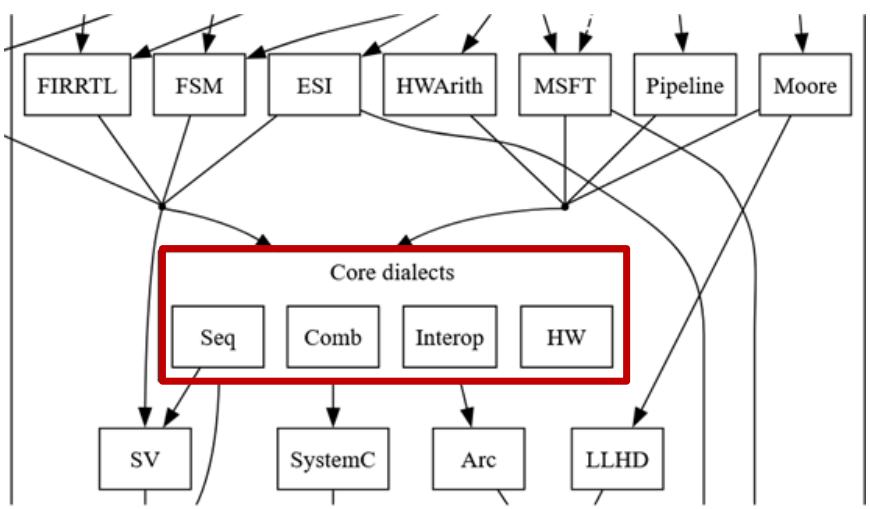
HW: structure (modules and ports)

Comb: Combinational logic (and, or, xor)

Seq: Sequential logic (registers)

Interop: between backends and tools






```
rtl.module @test1(%a: i12) -> (i32){
  %b = comb.add %a, %a : i12
  %c = comb.mul %a, %b : i12
  %d = comb.sext %c : (i12) -> i32
  rtl.output %d : i32
}
```

CIRCT: SV Dialect

Represents System Verilog syntax

Designed for export, not transformation

Readability and other syntactical improvements happen here

Guides towards compatibility, but offers escape hatch

```
rtl.module @sytemverilog(%clock: i1, %reset: i1) {
  %c0 = rtl.constant 0 : i32
  %c42 = rtl.constant 42 : i32
  %reg = sv.reg : !rtl.inout<i32>
  sv.alwaysff(posedge %clock) {
   sv.passign %reg, %c42 : i32
  } (asyncreset : posedge %reset) {
   sv.passign %reg, %c0 : i32
  }
}
```

https://circt.llvm.org/docs/Dialects/SV/RationaleSV/

CIRCT: Calyx Dialect

Components: the basic building block of Calyx programs, which has three sections: cells, wires, and control

Cells: instantiates sub-components like adders and registers

Assignments: The wires section, works like non-blocking assignments in RTL

Groups: a collection of assignments that are only active when run from the control program

Control: specifies the order to run groups

```
component main() -> () {
  cells {
    a = std_reg(32);
    b = std_reg(32);
  }
  wires {
    group write_a { ... }
    group read_a { ... }
    group write_b { ... }
  }
  control {
    seq {
      write_a;
      par {
        read_a;
        write_b;
      }
    }
  }
}
```

CIRCT: FSM Dialect

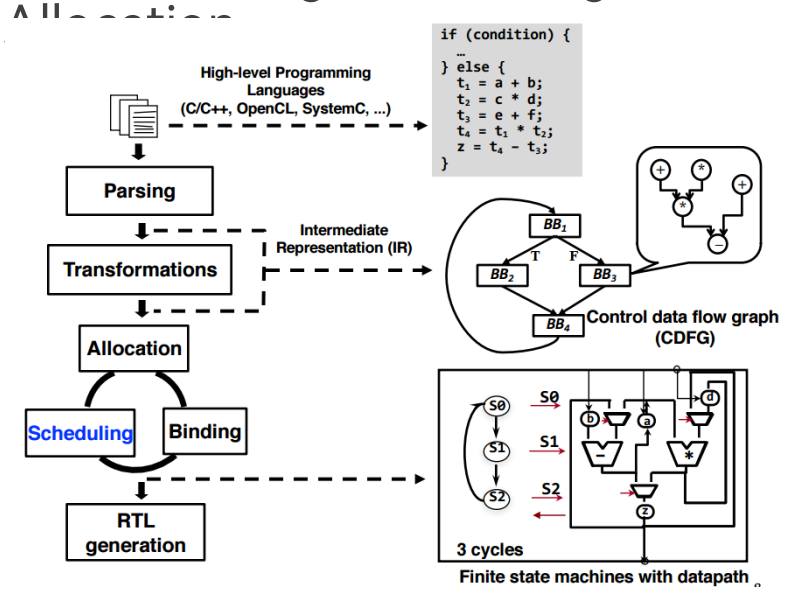
A set of abstractions for finite-state machine

- Provide explicit and structural representations of states, transitions, and internal variables of an FSM, allowing convenient analysis and transformation.
- Provide a target-agnostic representation of FSM, allowing the state machine to be instantiated and attached to other dialects from different domains.
- allow to lower the FSM abstraction into HW+Comb+SV (Hardware) and Standard+SCF+MemRef (Software) dialects for the purposes of simulation, code generation, etc.

```
fsm.machine @foo(%arg0: i1) -> i1 attributes {stateType = i1} {
  fsm.state "IDLE" output  {
    %true = constant true
    fsm.output %true : i1
  } transitions  {
    fsm.transition @BUSY ...
  }

  fsm.state "BUSY" output  {
    %false = constant false
    fsm.output %false : i1
  } transitions  {
    fsm.transition @BUSY ...
    fsm.transition @IDLE ...
  }
}
```

Key Algorithms in HLS

1) Scheduling; 2) Binding; 3) Allocation



Code repository Q&A

- "Chat with Copilot" in github.com
- deepwiki.com by Devin

an scheduling example in CIRCT

```
/// This class provides a framework to model certain scheduling problems as
/// lexico-parametric linear programs (LP), which are then solved with an
/// extended version of the dual simplex algorithm.
///
/// The approach is described in:
///  [1] B. D. de Dinechin, "Simplex Scheduling: More than Lifetime Sensitive
///      Instruction Scheduling", PRISM 1994.22, 1994.
///  [2] B. D. de Dinechin, "Fast Modulo Scheduling Under the Simplex Scheduling
```

```
std::unique_ptr<mlir::Pass> createMapArithToCombPass();
std::unique_ptr<mlir::Pass> createFlattenMemRefPass();
std::unique_ptr<mlir::Pass> createFlattenMemRefCallsPass();
std::unique_ptr<mlir::Pass> createStripDebugInfoWithPredPass(
    const std::function<bool(mlir::Location)> &pred);
std::unique_ptr<mlir::Pass> createMaximizeSSAPass();
std::unique_ptr<mlir::Pass> createInsertMergeBlocksPass();
std::unique_ptr<mlir::Pass> createPrintOpCountPass();
std::unique_ptr<mlir::Pass>
createMemoryBankingPass(ArrayRef<unsigned> bankingFactors = {},
                        ArrayRef<unsigned> bankingDimensions = {});
std::unique_ptr<mlir::Pass> createIndexSwitchToIfPass();
std::unique_ptr<mlir::Pass> createHierarchicalRunner(
    const std::string &topName,
    llvm::function_ref<void(mlir::OpPassManager &)> pipeline,
    bool includeBoundInstances = false);
```

memory banking (kind of resource alloc.) in CIRCT

Short Summary of the CIRCT Project

CIRCT project is “codified” digital design experiences

“Large Circuit Model”: learned design experiences?

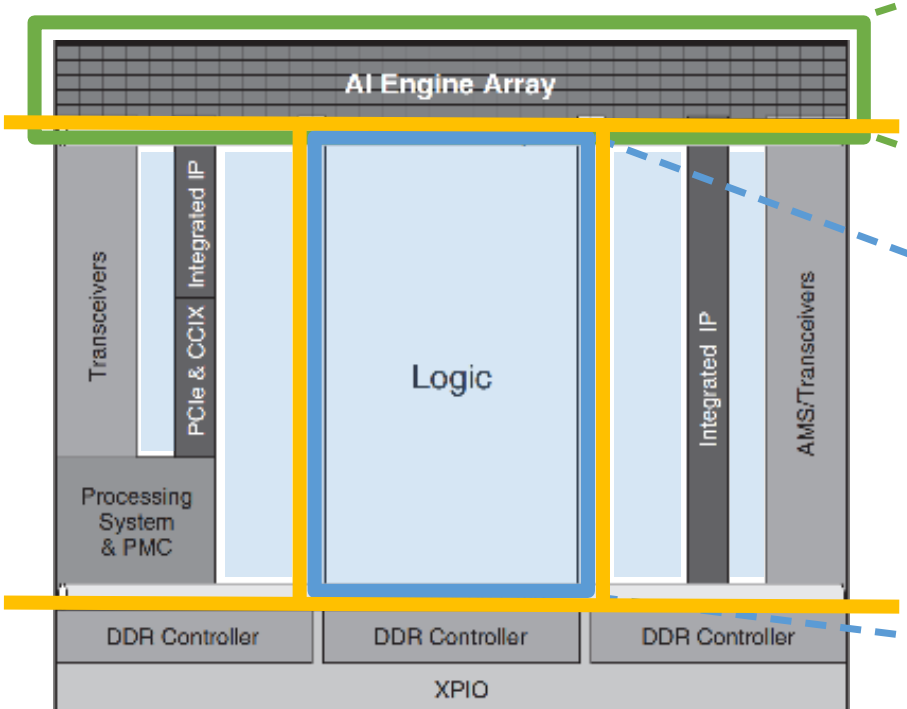
- L. Chen et al., “Large circuit models: opportunities and challenges,” SCIS vol. 67, no. 10, Oct. 2024.

Experiences in “good-for-hardware” patterns

James C. Hoe, "Lecture 6: Good-for-HW Computation Models," CMU ECE 643, Fall 2023.

- FSM-Datapath
- Systolic Array
- Data Parallel
- Dataflow
- Stream Processing

可重构架构的演化 FPGA+CGRA



a modern “FPGA” architecture (AMD Versal)

NoC

coarse grain

* word-level network of VLIW PEs
* spatial & temporal

fine grain

* bit-level network of LUTs and FFs
* combinational & sequential

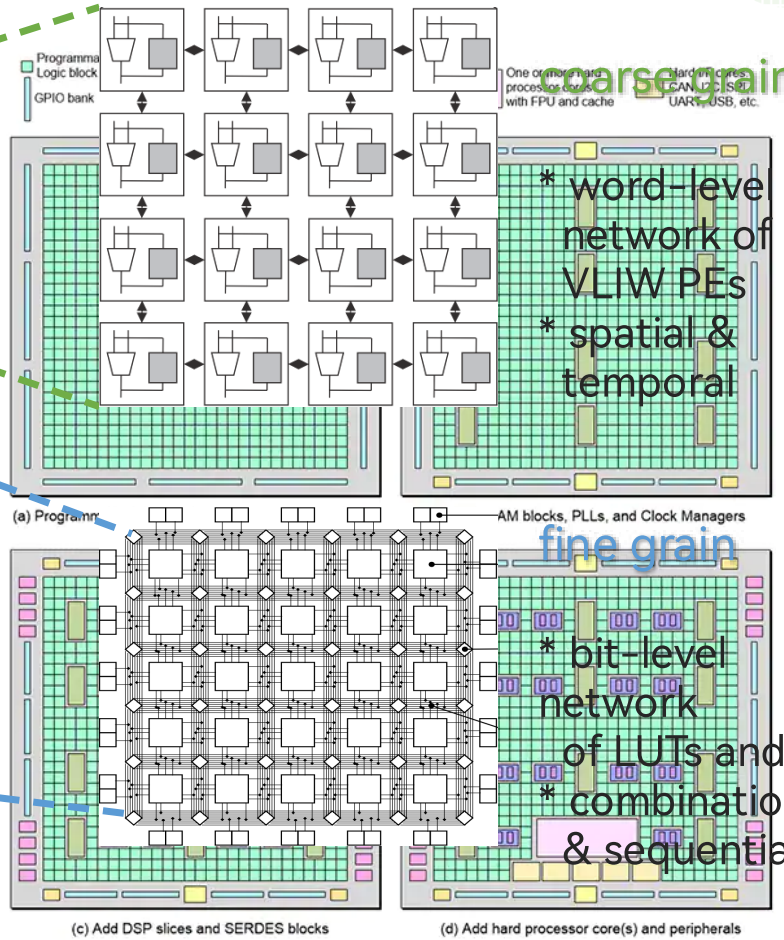
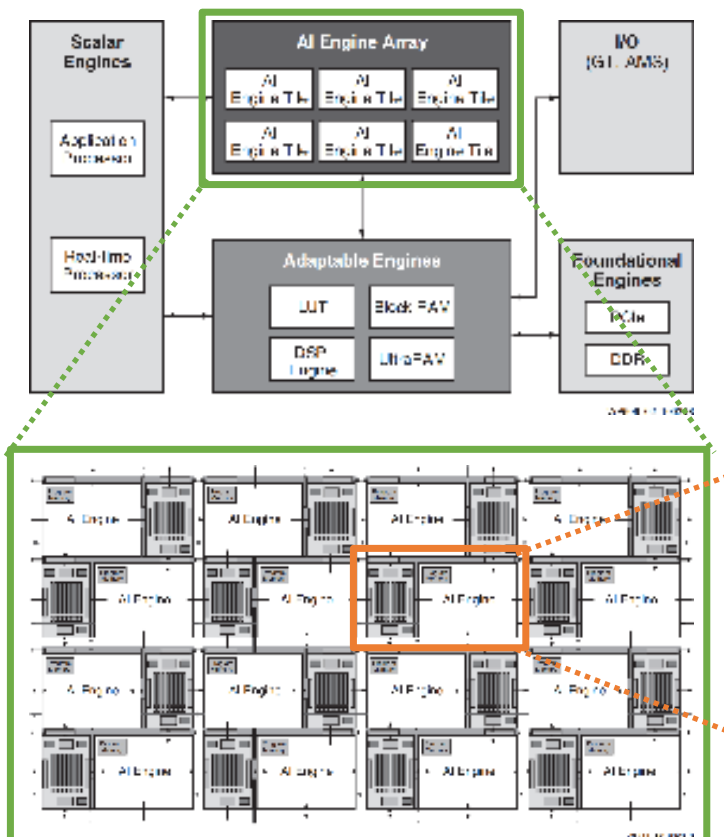
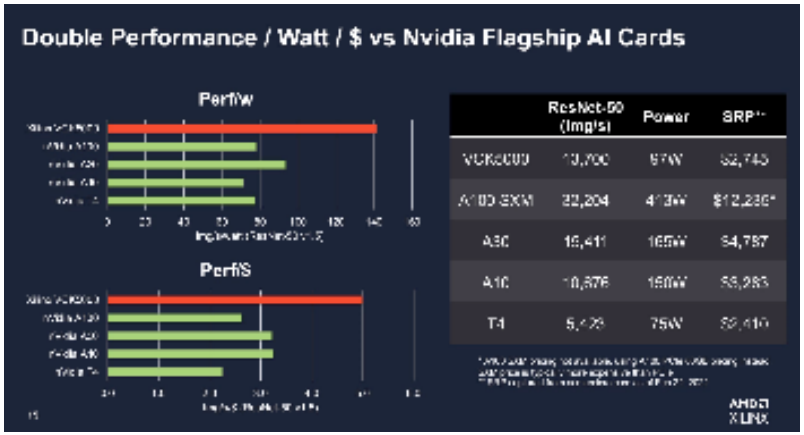


image source: digikey.com, Xilinx WP506

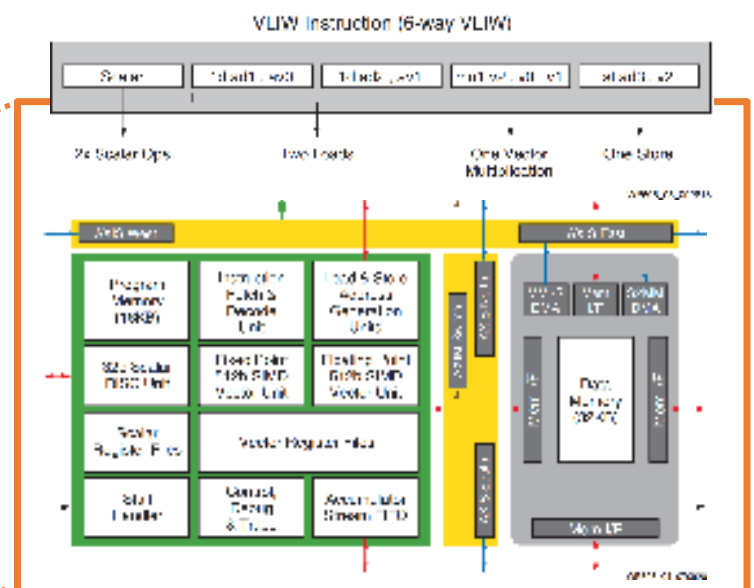
现代可重构计算的微架构

现代 PE 阵列架构的 VLIW 单元

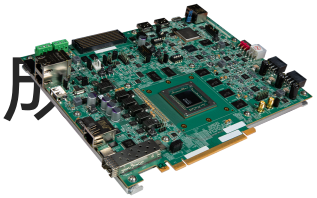


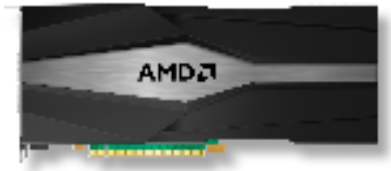


ResNet-50 推理任务：现代可重构架构 (如 AMD Versal) 的能效和性价比均优于 A100

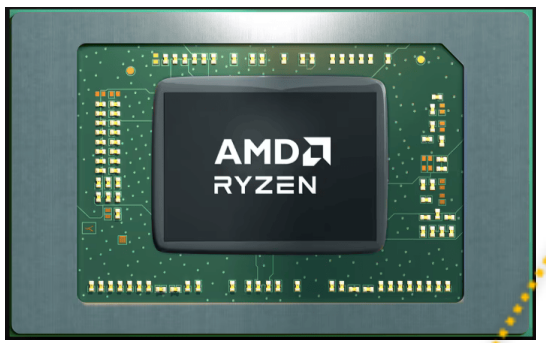


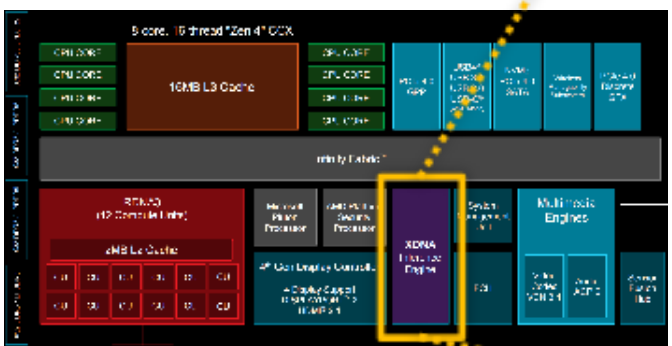
AI 引擎(AIE, AI Engine) 在 CPU 和 FPGA 的集成



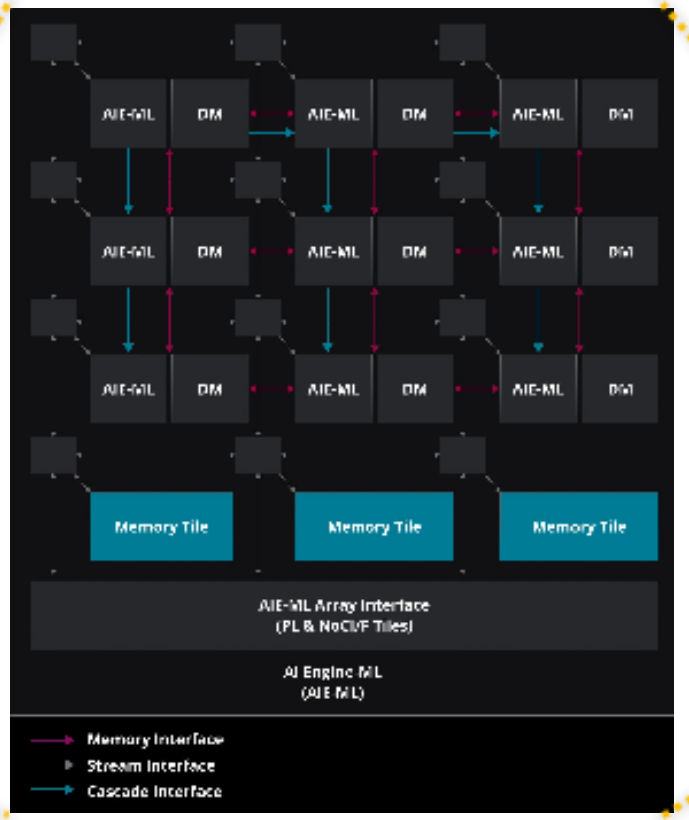


AMD Ryzen AI series
(CPU + AIE)

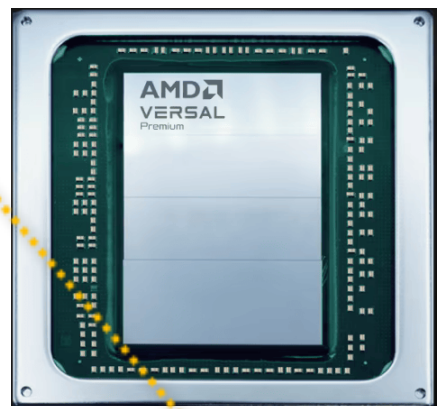


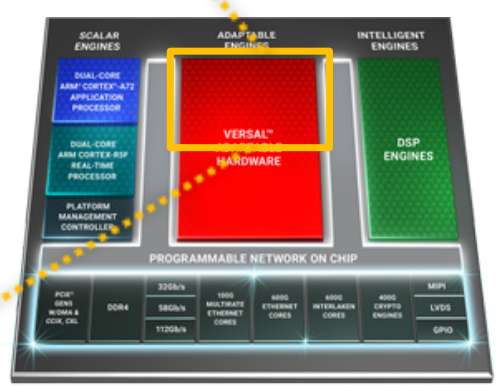


AI Engine 阵列



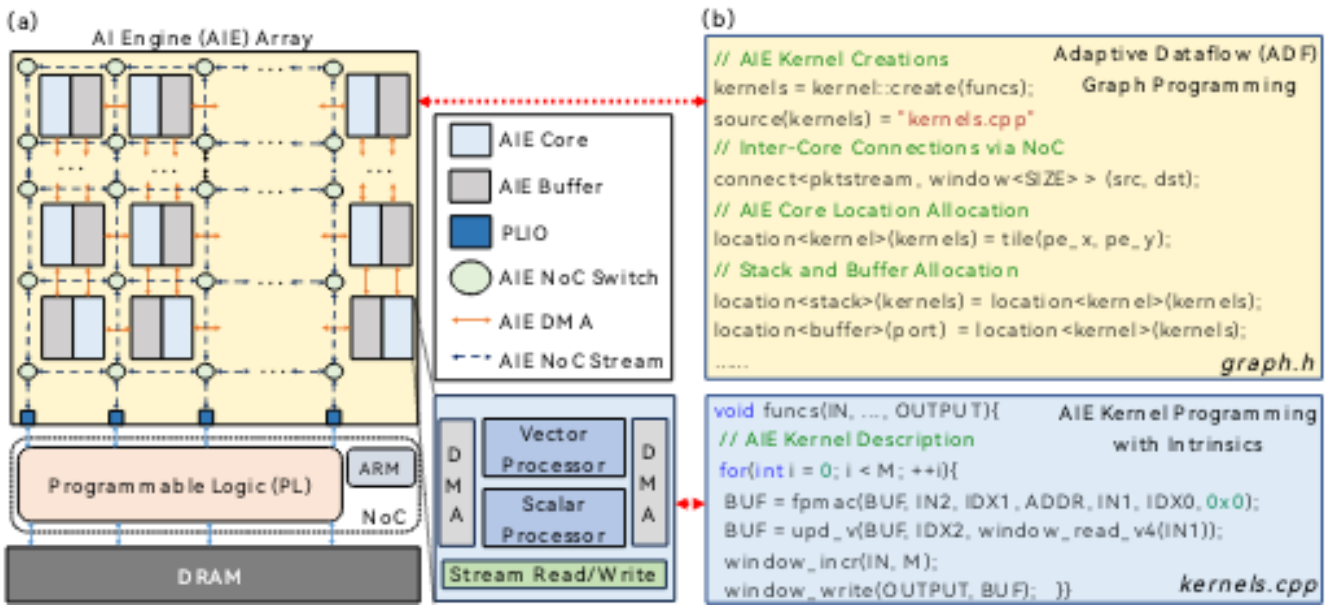
AMD Versal VU9P
(FPGA + AIE)



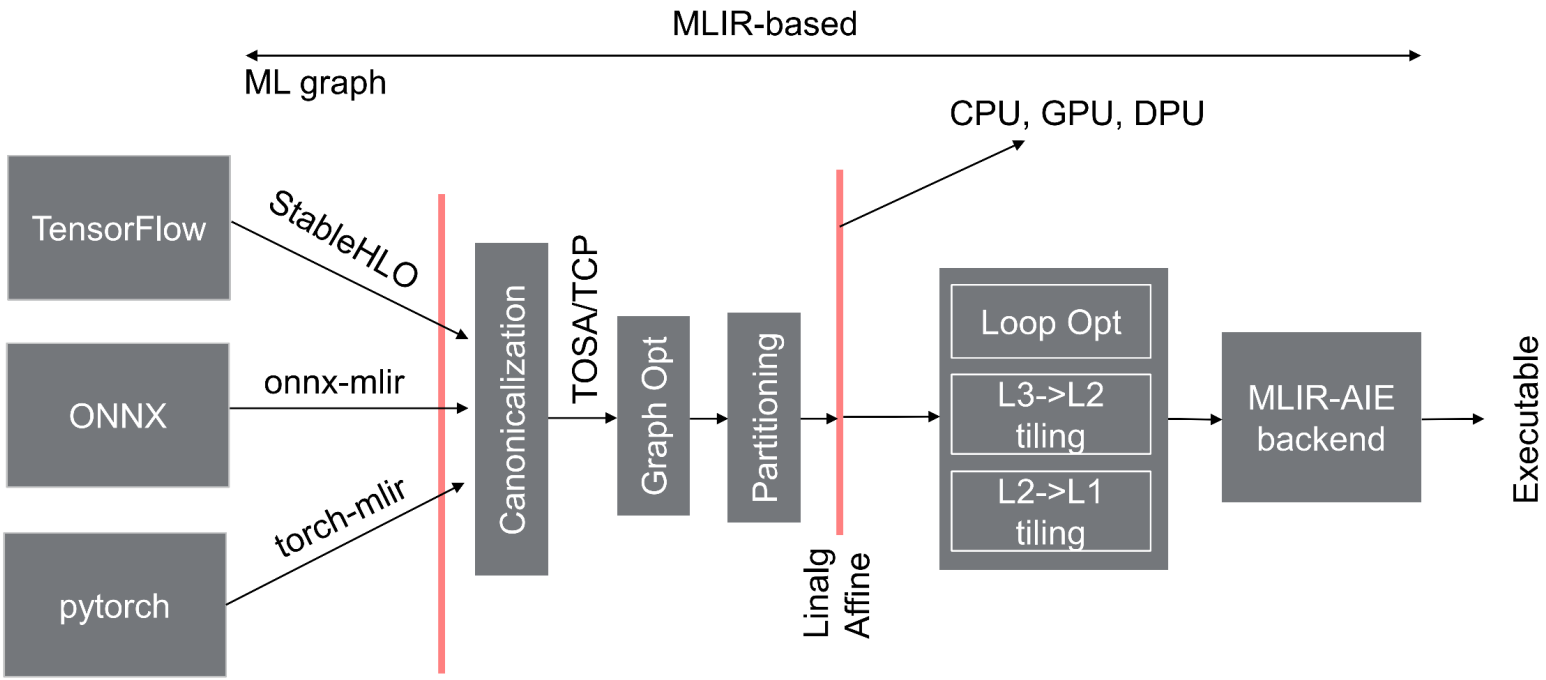


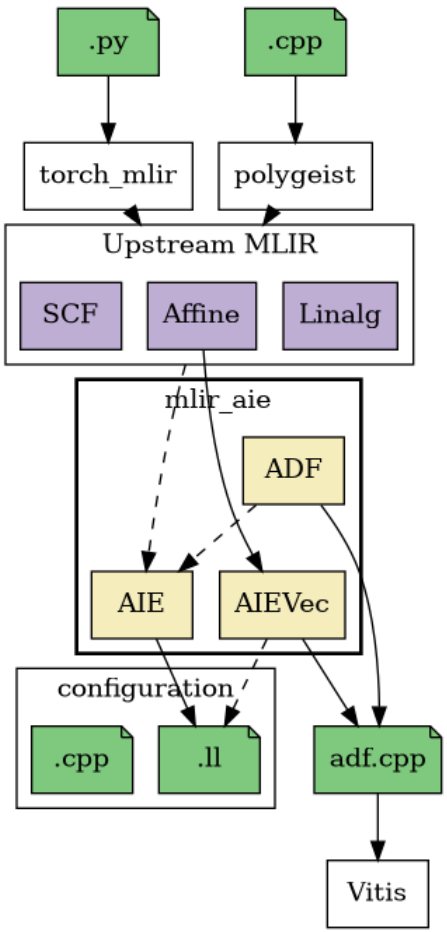
ACAP架构应用的编程模型

硬件结构	软件编程模型
AIE 阵列	ADF 图描述
AI Engine	AIE 计算核代码



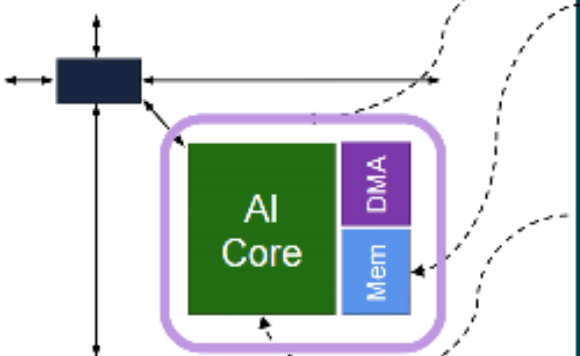
MLIR-AIE





Stephen Neuendorffer, Kristof Denolf, Erwei Wang, Jack Lo and Andra Bisca, "Leveraging MLIR to Design for AI Engines," FCCM Tutorial 2023.

Running Code on a Core

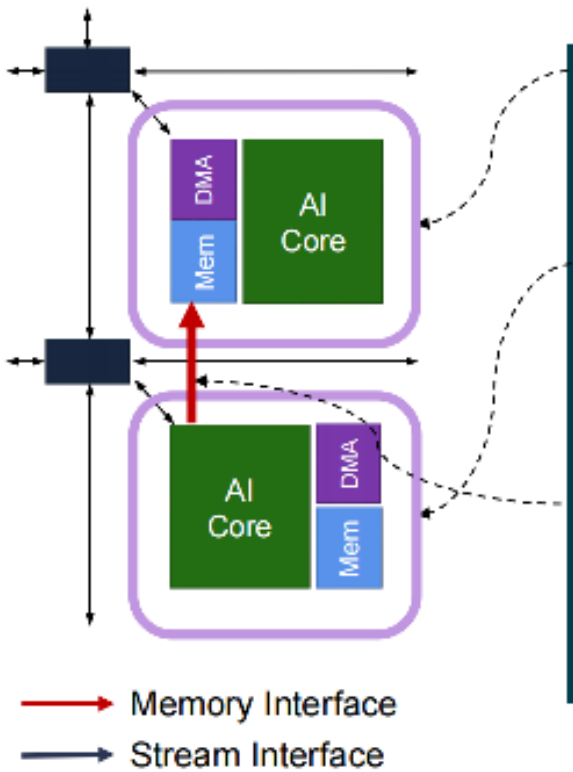



```
%tile13 = AIE.tile(1, 3)
%buf13_0 = AIE.buffer(%tile13)
        { sym_name = "a" } : memref<256xi32>

%core13 = AIE.core(%tile13) {
  %val1 = constant 7 : i32
  %idx1 = constant 3 : index
  %2 = addi %val1, %val1 : i32
  memref.store %2, %buf13_0[%idx1] : memref<256xi32>

  AIE.end
}
```

Memory Interface
Stream Interface

Stephen Neuendorffer, Kristof Denolf, Erwei Wang, Jack Lo and Andra Bisca, "Leveraging MLIR to Design for AI Engines," FCCM Tutorial 2023.

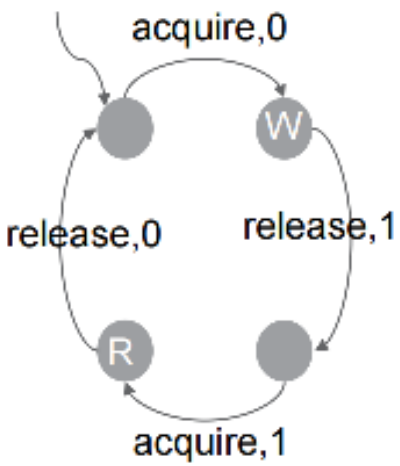
Moving Buffers




```
%tile14 = AIE.tile(1, 4)
%buf = AIE.buffer(%tile14)
        { sym_name = "a" } : memref<256xi32>

%tile13 = AIE.tile(1, 3)
%core13 = AIE.core(%tile13) {
  %val1 = constant 7 : i32
  %idx1 = constant 3 : index
  %2 = addi %val1, %val1 : i32

  memref.store %2, %buf[%idx1] : memref<256xi32>

  AIE.end
}
```


Stephen Neuendorffer, Kristof Denolf, Erwei Wang, Jack Lo and Andra Bisca, "Leveraging MLIR to Design for AI Engines," FCCM Tutorial 2023.

Synchronization with Locks

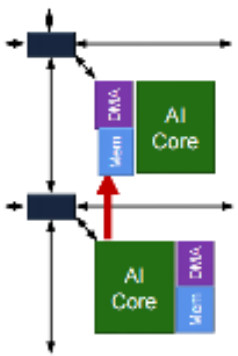


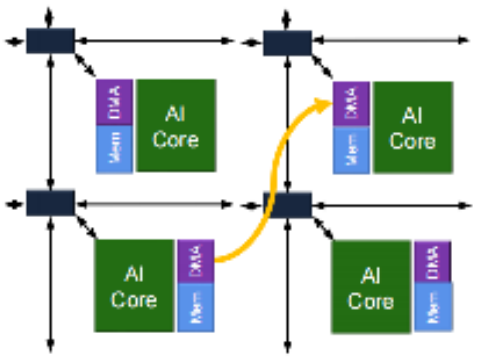

```
%tile14 = AIE.tile(1, 4)
%lock = AIE.lock(%tile14) { sym_name = "a_lock" }
%buf = AIE.buffer(%tile14)
       { sym_name = "a" } : memref<512xi32>

%tile13 = AIE.tile(1, 3)
%core13 = AIE.core(%tile13) {
  AIE.useLock(%lock, "Acquire", 0)
  memref.store %2, %buf[%idx1] : memref<512xi32>
  AIE.useLock(%lock, "Release", 1)
}

%core14 = AIE.core(%tile14) {
  AIE.useLock(%lock, "Acquire", 1)
  %data = memref.load %buf[%idx1] : memref<512xi32>
  AIE.useLock(%lock, "Release", 0)
}
```


Stephen Neuendorffer, Kristof Denolf, Erwei Wang, Jack Lo and Andra Bisca, "Leveraging MLIR to Design for AI Engines," FCCM Tutorial 2023.

Object FIFO Lowers to Communication Components Based on the Endpoints

aie-opt --aie-objectFifo-stateful-transform



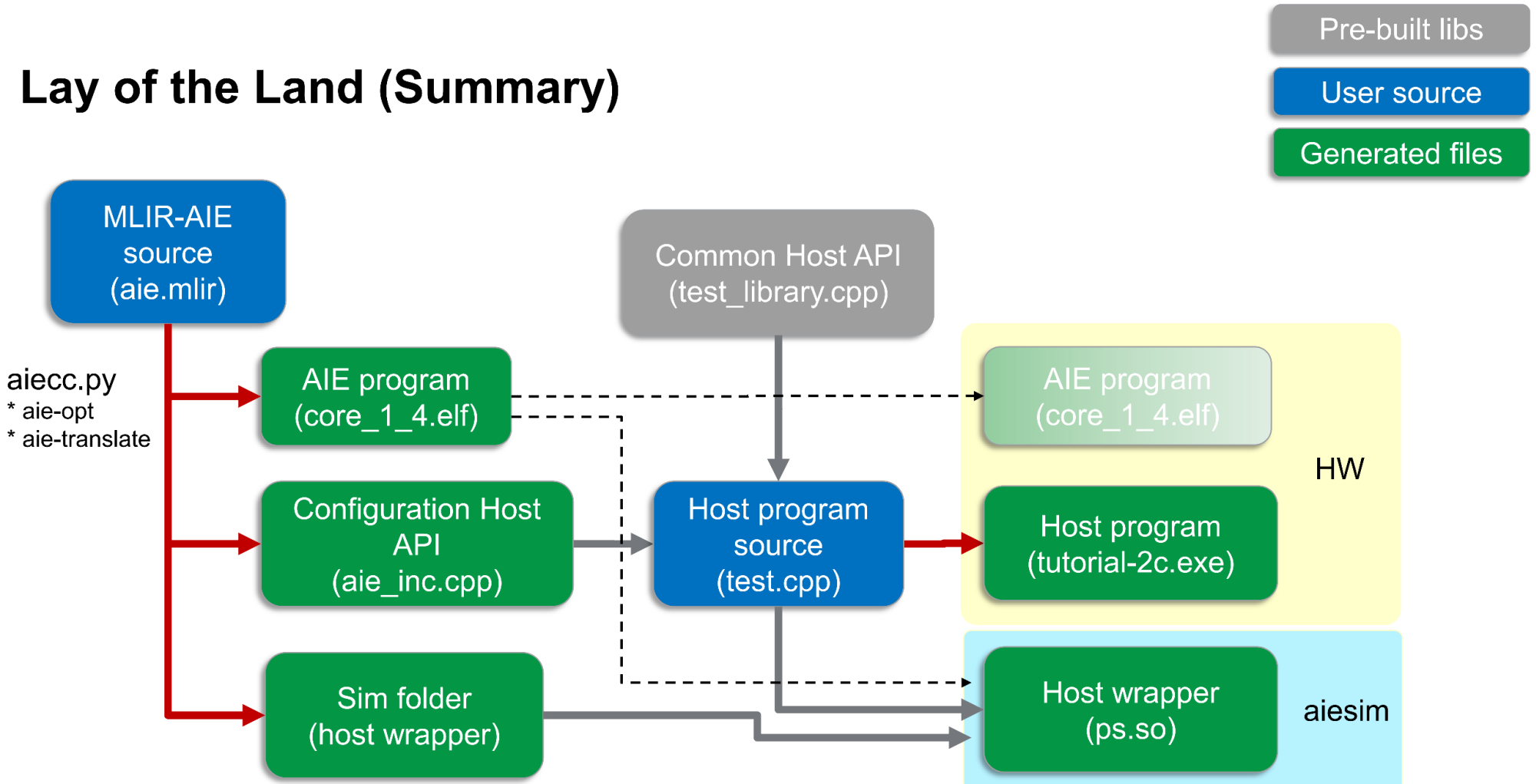



```
%4 = AIE.buffer(%0) : memref<16xi32>
%5 = AIE.lock(%0, 0)
%6 = AIE.buffer(%0) : memref<16xi32>
%7 = AIE.lock(%0, 1)
%8 = AIE.core(%0) { ... }
%9 = AIE.core(%1) { ... }
```

```
%6 = AIE.buffer(%0) : memref<16xi32>
%7 = AIE.lock(%0, 0)
%8 = AIE.buffer(%0) : memref<16xi32>
%9 = AIE.lock(%0, 1)
%10 = AIE.buffer(%2) : memref<16xi32>
%11 = AIE.lock(%2, 1)
%12 = AIE.buffer(%2) : memref<16xi32>
%13 = AIE.lock(%2, 2)
%14 = AIE.core(%0) { ... }
%15 = AIE.core(%2) { ... }
%16 = AIE.mem(%0) { ... }
%17 = AIE.mem(%2) { ... }
AIE.flow(%0, DMA : 0, %1, DMA : 0)
```


Stephen Neuendorffer, Kristof Denolf, Erwei Wang, Jack Lo and Andra Bisca, "Leveraging MLIR to Design for AI Engines," FCCM Tutorial 2023.

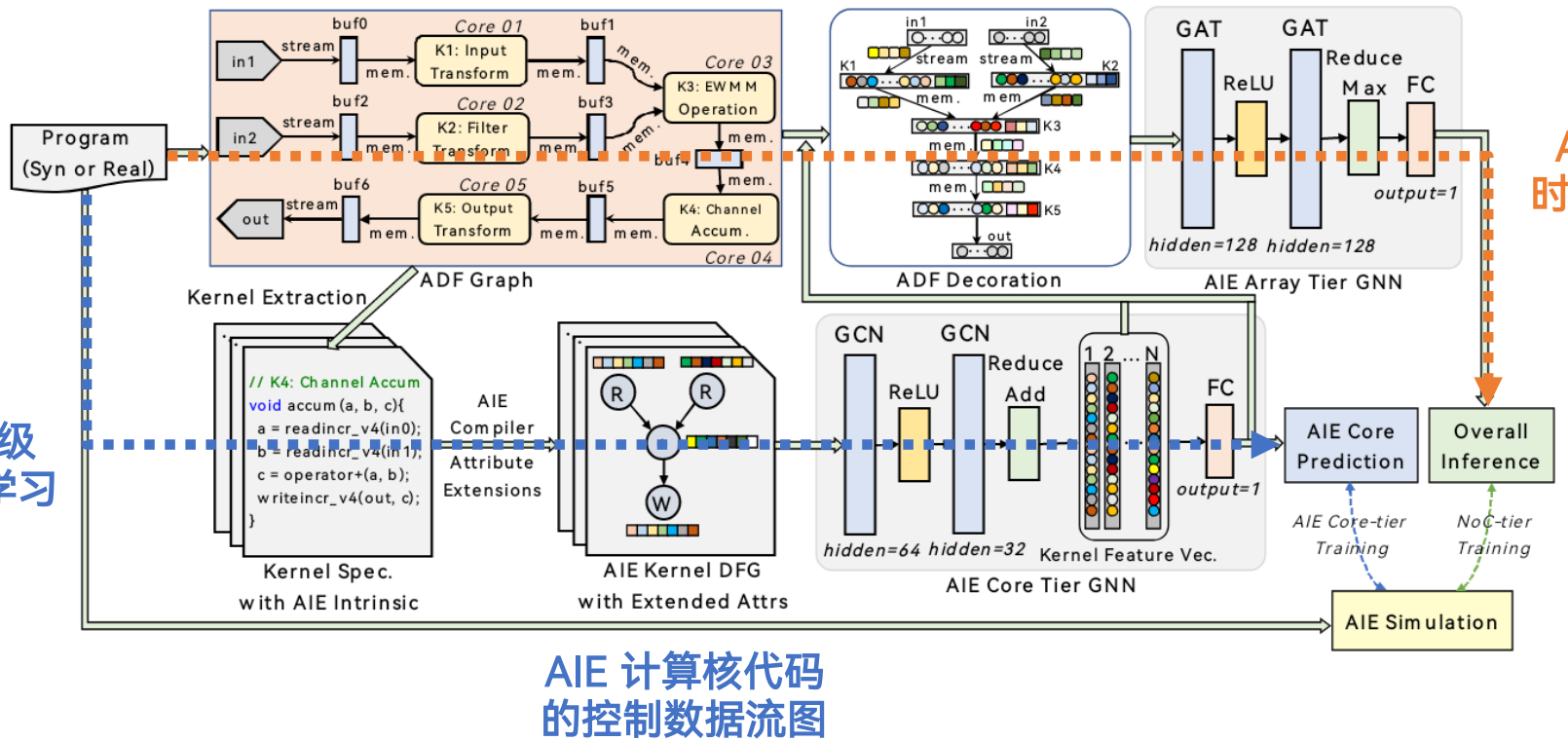
Lay of the Land (Summary)



Stephen Neuendorffer, Kristof Denolf, Erwei Wang, Jack Lo and Andra Bisca, "Leveraging MLIR to Design for AI Engines," FCCM Tutorial 2023.

相关工作：G2PM 基于双层图学习的性能模型

AIE 阵列间通信
的数据流图



Tuo Dai, Bizhao Shi, Guojie Luo, "G2PM: Performance Modeling for ACAP Architecture with Dual-Tiered Graph Representation Learning," DAC 2024.

相关工作：WideSA - AIE阵列映射器（结果节选）

整体性能比较：达到100% AIE 利用率和36.02 TOPS 峰值性能

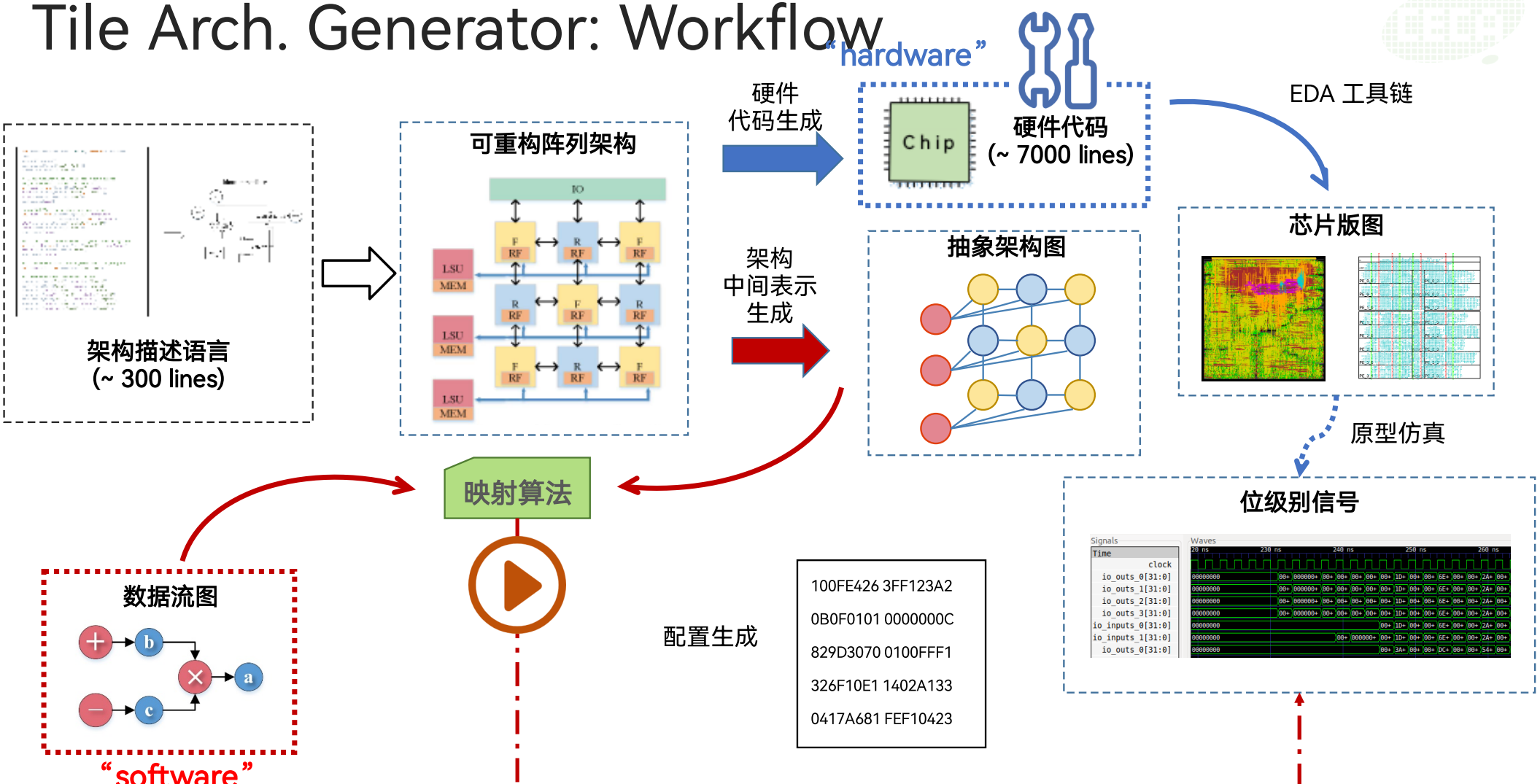
方法	指标	MM				2D-Conv				2D-FFT		FIR Filter			
数据类型		Float	Int8	Int16	Int32	Float	Int8	Int16	Int32	Cfloat	Cint16	Float	Int8	Int16	Cfloat
基线	#AIEs	384	384	384	384	-	256	-	-	10	10	10	10	10	10
	TOPS	3.73	29.78	7.82	3.72	-	31.40	-	-	0.04	0.13	0.15	2.56	0.62	0.15
	TOPS#AIEs	0.010	0.077	0.020	0.010	-	**0.123**	-	-	**0.004**	**0.013**	**0.015**	**0.256**	**0.062**	**0.015**
WideSA	#AIEs	400	400	400	400	400	400	400	400	320	320	256	256	256	256
	TOPS	**4.15**	**32.49**	**8.10**	**3.92**	**4.50**	**36.02**	**10.35**	**4.48**	**1.10**	**3.83**	**2.92**	**39.3**	**9.47**	**2.89**
	TOPS/#AIEs	**0.010**	**0.081**	**0.020**	**0.010**	**0.011**	0.090	**0.025**	**0.011**	0.003	0.012	0.012	0.100	0.037	0.011

矩阵乘法与常规 FPGA 加速器比较：2.25倍的能效比提升

	PL-only				WideSA			
数据类型	Float	Int8	Int16	Int32	Float	Int8	Int16	Int32
DSPs	1536	1528	1516	1536	152	60	67	65
#AIEs	0	0	0	0	400	400	400	400
吞吐量	0.59	5.77	2.16	0.60	**4.15**	**32.49**	**8.10**	**3.92**
功率 (W)	19.5	18.8	18.6	19.5	55.8	54.4	54.9	55.6
能效	0.03	0.31	0.12	0.03	0.07	0.60	0.15	0.07
能效比	1×	1×	1×	1×	**2.25×**	**1.94×**	**1.29×**	**2.25×**

Tuo Dai, Bizhao Shi, Guojie Luo, "WideSA: A High Array Utilization Mapping Scheme for Uniform Recurrences on ACAP," DATE 2024.

Tile Arch. Generator: Workflow



https://github.com/pku-dasys/pillars

总结：CIRCT 和 MLIR-AIE

CIRCT: Circuit IR Compilers and Tools

- https://circt.llvm.org/
- The CIRCT project is an (experimental!) effort looking to apply MLIR and the LLVM development methodology to the domain of hardware design tools.

MLIR-AIE: Toolchain for AMD AI Engine devices

- https://xilinx.github.io/mlir-aie/
- MLIR-AIE is a toolchain providing low-level device configuration for Versal AIEngine-based devices. Support is provided to target the AIEngine portion of the device, including processors, stream switches, TileDMA and ShimDMA blocks. Backend code generation is included, targetting the LibXAIE library, along with some higher-level abstractions enabling higher-level design.