Progressive Arithmetic Lowering from Tensor Kernels to Synthesizable Datapaths

MLIR Workshop @ Euro LLVM Developers' Meeting

13 April 2026

Inria
INSA Institut National des Sciences Appliquées Lyon
CITI Lab
anr agence nationale de la recherche

Louis Ledoux
Pierre Cochard
Florent de Dinechin

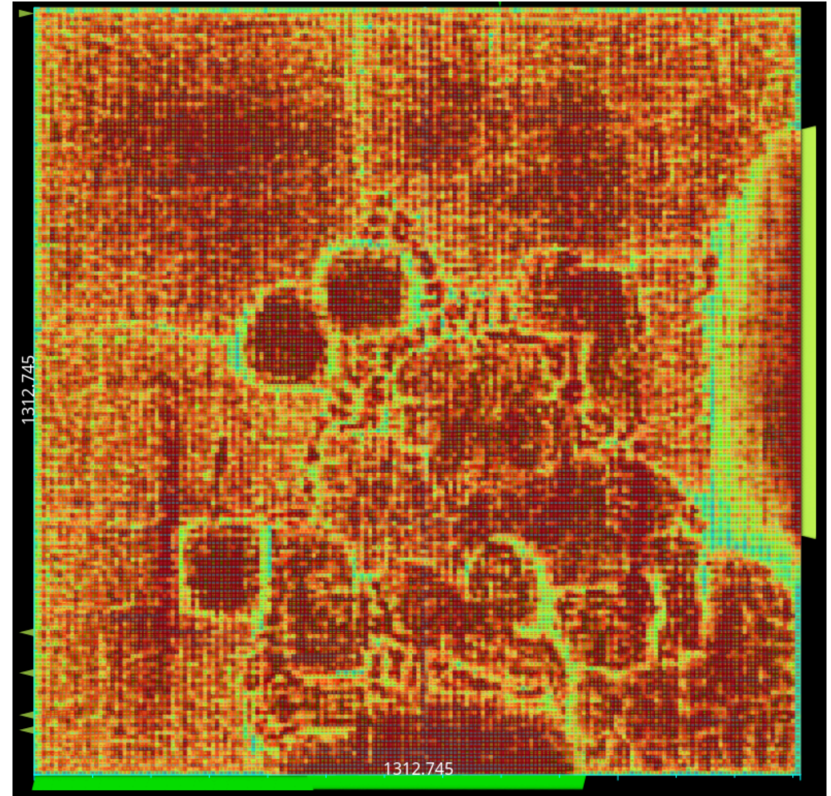
0.1 Motivation AI

Llama.py Python

```python
class LlamaFfnSublayer(nn.Module):
    """Llama FFN sublayer using SwiGLU."""

    def __init__(
        self,
        dim: int = 512,
        hidden_dim: int | None = None,
        multiple_of: int = 256,
    ):
        super().__init__()
        if hidden_dim is None:
            hidden_dim = 4 * dim
            hidden_dim = int(2 * hidden_dim / 3)
            hidden_dim = multiple_of * (
                (hidden_dim + multiple_of - 1)
            )
        self.w_gate = nn.Linear(dim, hidden_dim, bias=False)
        self.w_up = nn.Linear(dim, hidden_dim, bias=False)
        self.w_down = nn.Linear(hidden_dim, dim, bias=False)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        gate = F.silu(self.w_gate(x))
        up = self.w_up(x)
        return self.w_down(gate * up)
```

MatMul projections

SiLU + elementwise mul

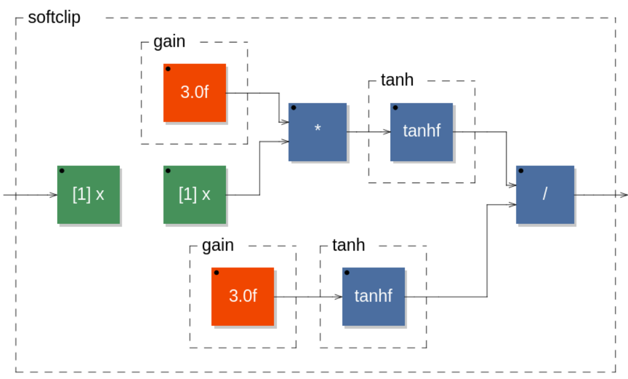


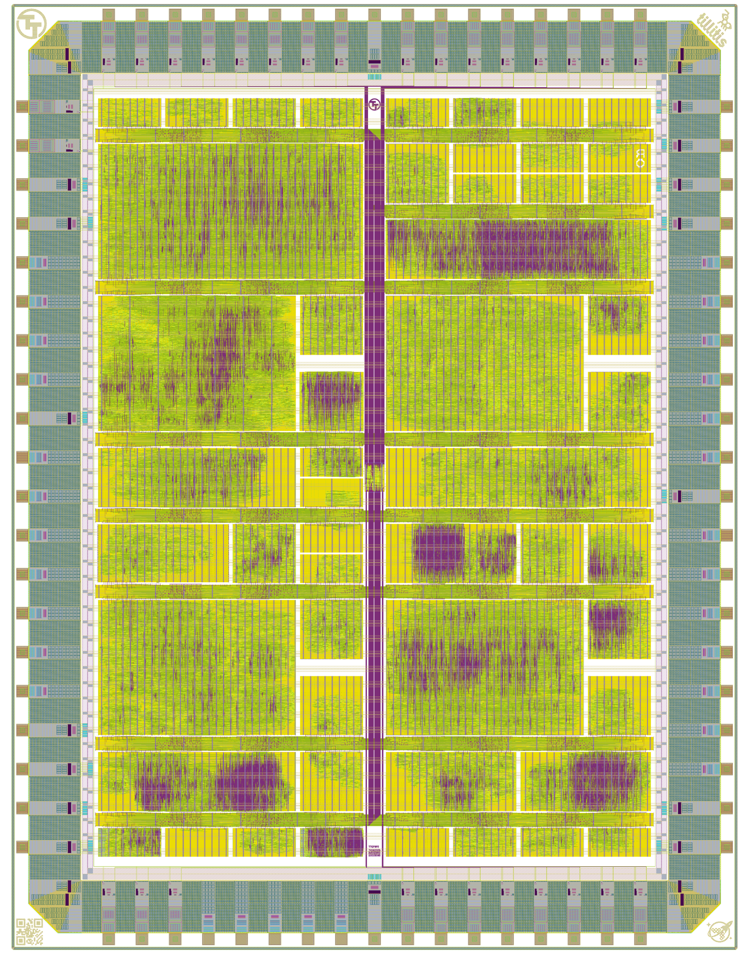
Llama ASIC routing

0.2 Motivation Audio (DSP)

softclip.dsp Faust

```
// Soft clipping waveshaper.
// Hyperbolic tangent via foreign function.

gain = 3.0;

tanh = ffunction(float tanhf|tanh|tanhl (float), <math.h>,
 "");

softclip(x) = tanh(gain * x) / tanh(gain);

process = _,_ : par(i, 2, softclip);
```



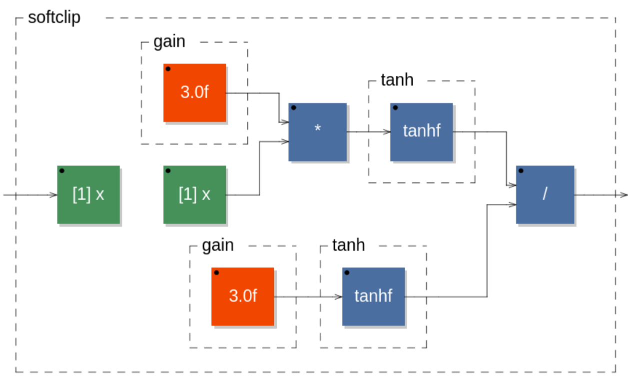


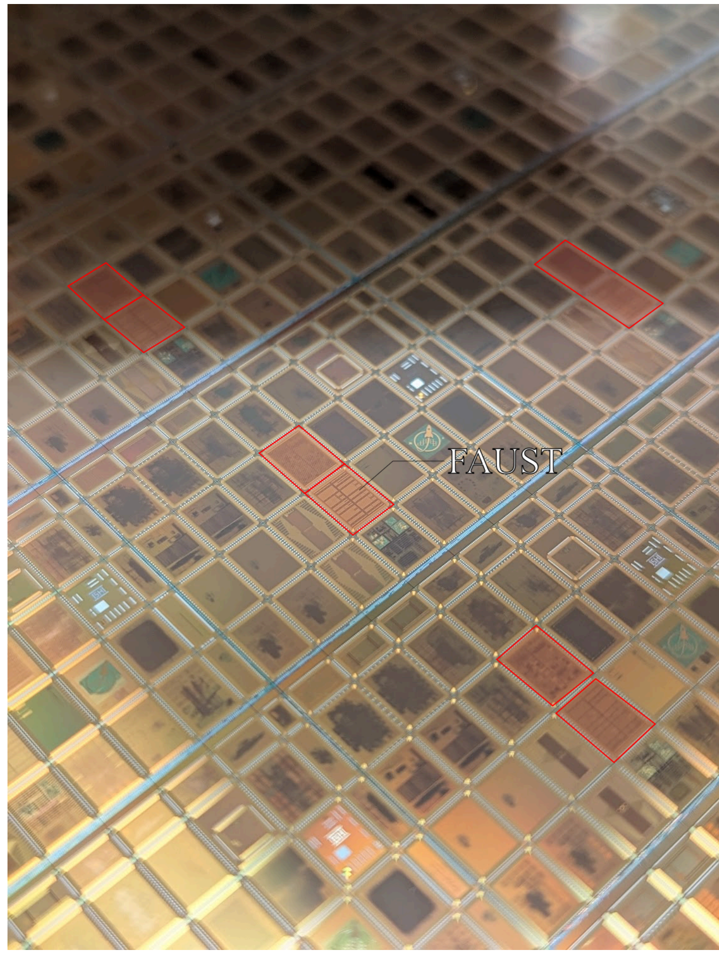
TinyTapeout GF0P02 wafer.space

0.2 Motivation Audio (DSP)

softclip.dsp Faust

```
// Soft clipping waveshaper.
// Hyperbolic tangent via foreign function.

gain = 3.0;

tanh = ffunction(float tanhf|tanh|tanhl (float), <math.h>,
 "");

softclip(x) = tanh(gain * x) / tanh(gain);

process = _,_ : par(i, 2, softclip);
```



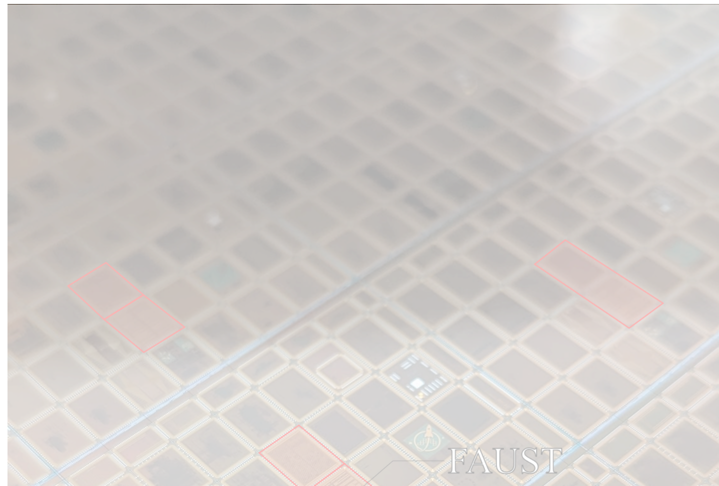


TinyTapeout GF0P02 wafer.space

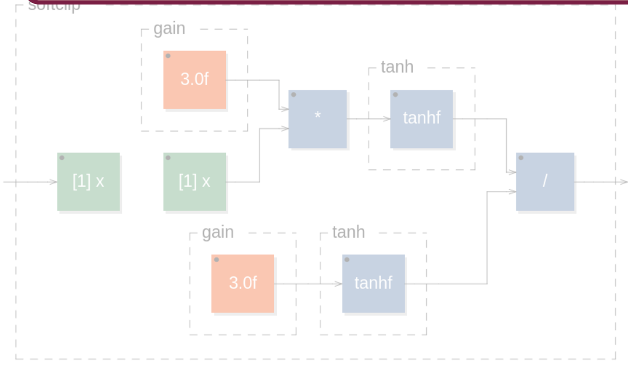
0.2 Motivation Audio (DSP)

softclip.dsp Faust

```
// Soft clipping waveshaper.
// Hyperbolic tangent via foreign function.

gain = 3.0;

tanh = ffunction(float tanhf|tanh|tanhl (float), <math.h>,
 "");

softclip(x) = tanh(gain * x) / tanh(gain);

process =      : par(i, 2, softclip);
```



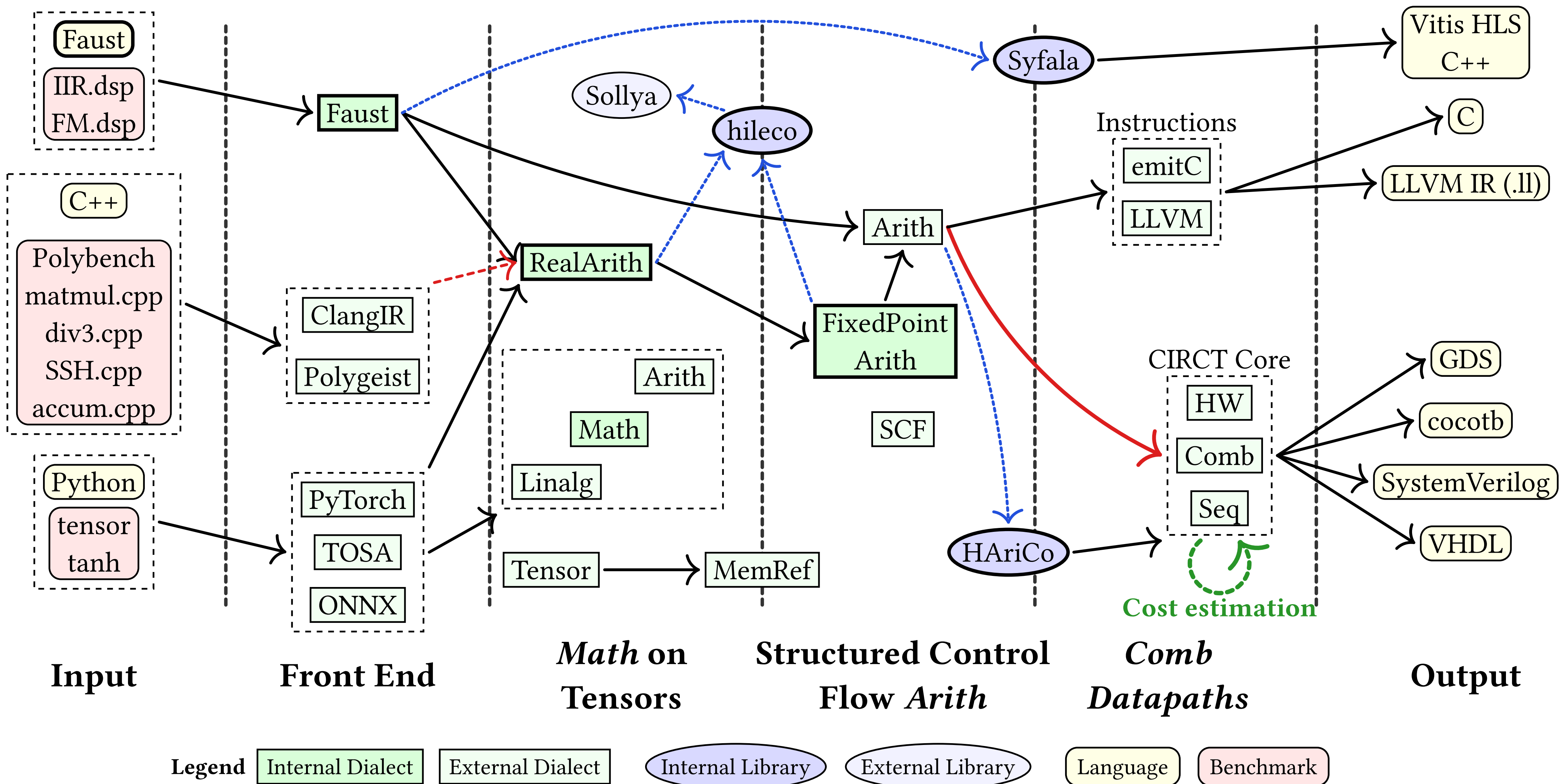
Problem to Silicon ⇒ In reality… multi intermediate arithmetic levels.



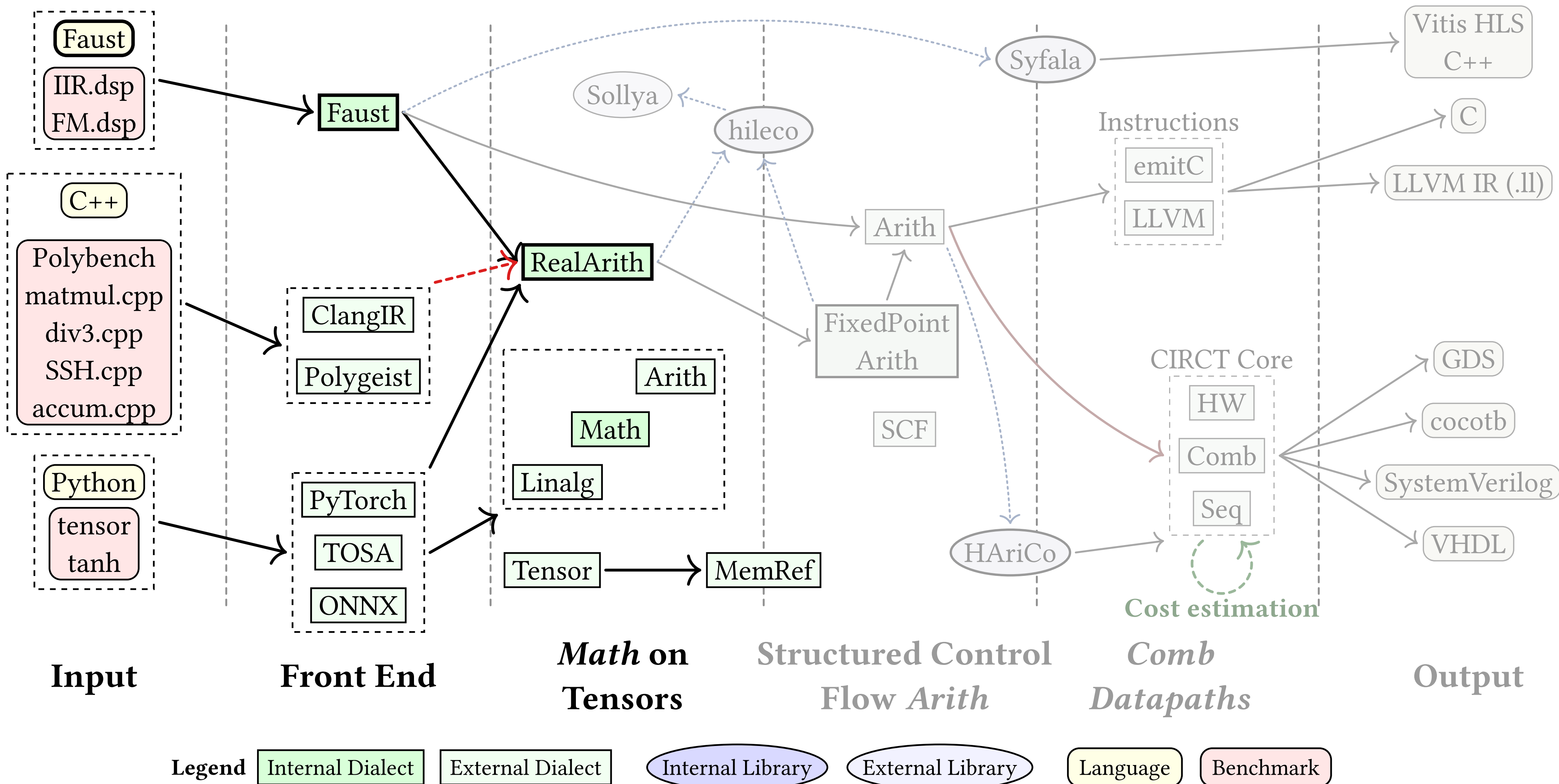
TinyTapeout GF0P02 wafer.space

1. Emeraude-MLIR

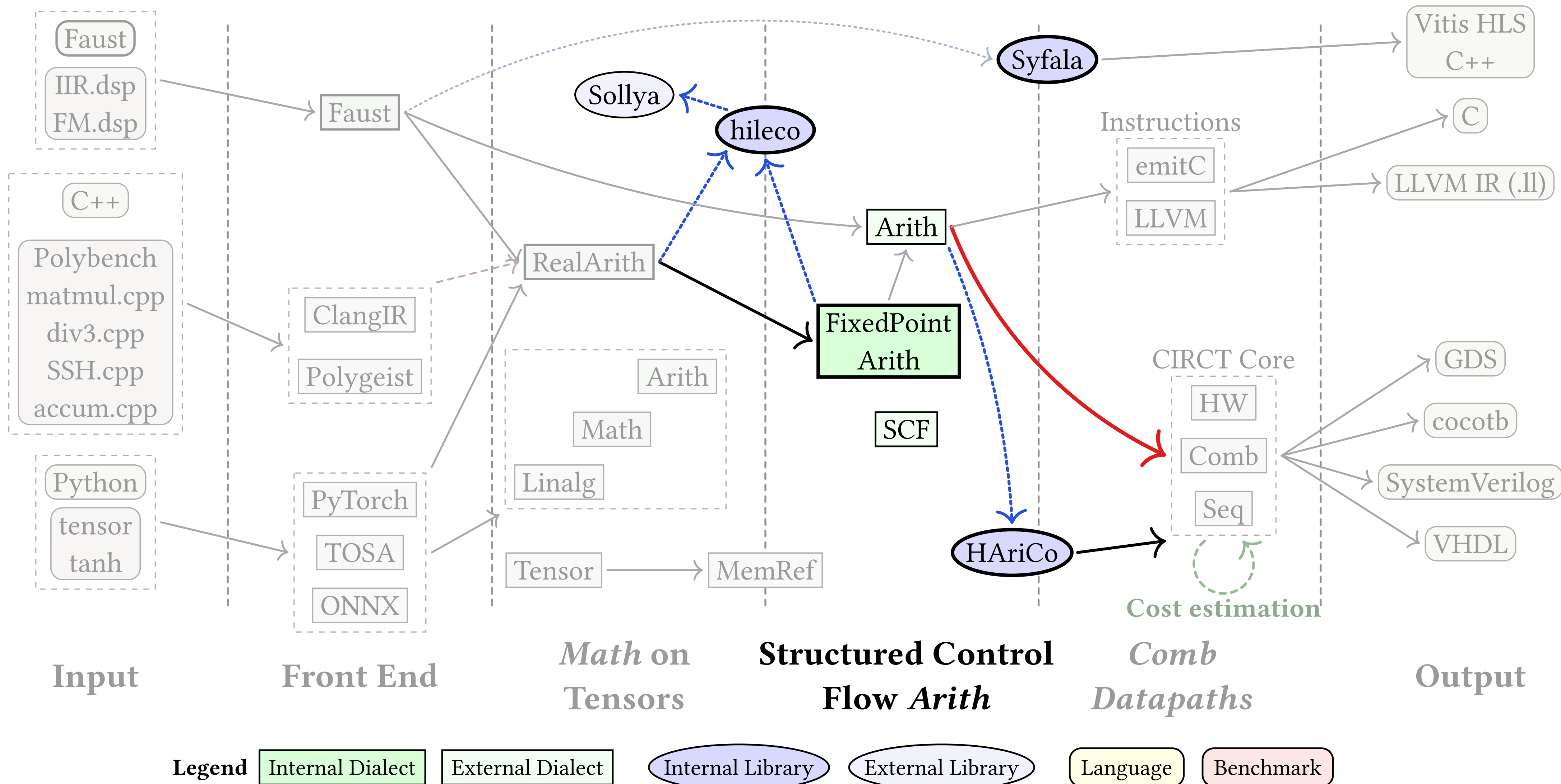
1.1 Our take on those end-to-end levels



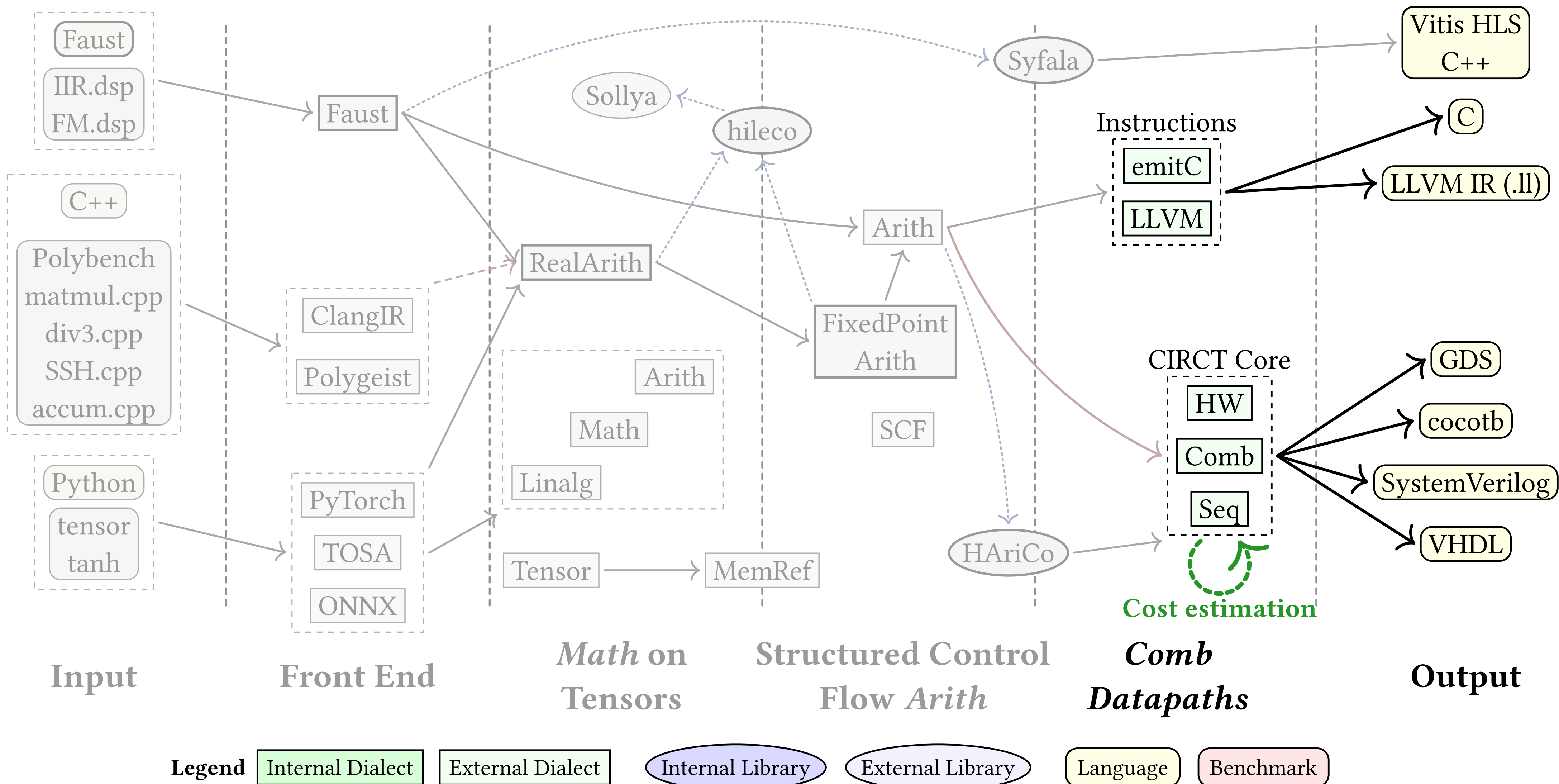
1.1 Our take on those end-to-end levels



1.1 Our take on those end-to-end levels



1.1 Our take on those end-to-end levels



2. Motivations at Both ends: Frontends & Backends

2.1 Frontends on one hand (end)

Faust / DSP

- **FIR / IIR recurrences** on sample streams
- **Fixed-point friendly** audio kernels
- **Nonlinear audio functions:** saturation, waveshaping, modal/finite-difference-time-domain physical modelling
- **Precision control:** range, quantization, overflow

Tensor / ML

- **Tensor contractions:** (`matmul`, batched GEMM)
- **Nonlinear kernels:** SiLU / exp / sigmoid / softmax
- **Accumulation depth:** rounding-error growth
- **Precision control:** mixed-precision per layer

2.1 Frontends on one hand (end)

Faust / DSP

- **FIR / IIR recurrences** on sample streams
- **Fixed-point friendly** audio kernels
- **Nonlinear audio functions:** saturation, waveshaping… domain phys…
- **Precision control:** range, quantization, overflow

Tensor / ML

- **Tensor contractions:** (`matmul`, batched GEMM)
- **Nonlinear kernels:** SiLU / exp / sigmoid /
- …ding-error growth
- **Precision control:** mixed-precision per layer

Arithmetic motivations of different natures

2.2 Backends on the other hand (end)

PPA objectives: area, power, performance… … or budgets / constraints

FPGA

- Resource budgets: LUT, FF, BRAM, DSP
- Reconfigurable
- Throughput-oriented

ASIC

- Standard cells (Resizing)
- Process Design Kit
  - 7nm, 180nm, corners, high density
- Physical constraints: congestion, wiring
- DRC/LVS and signoff requirements

2.2 Backends on the other hand (end)

PPA objectives: area, power, performance… … or budgets / constraints

FPGA

- Resource budgets: LUT, FF, BRAM, DSP
- Reconfigurable
- Throughput-o

ASIC

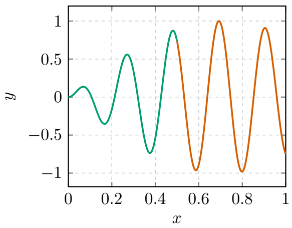
- Standard cells (Resizing)
- Process Design Kit
- h density
- tion, wiring
- DRC/LVS and signoff requirements

Hardware constraints of different natures

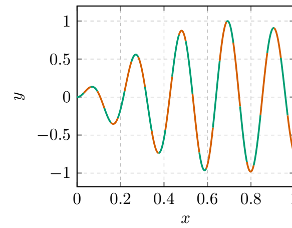
3. High-Level *Math* on *Tensors*

3.1 Polynomial Approximations

- **`RealArith`** and **`FixedPointArith`**.
- `--realarith-to-fixed_pt_arith` for polynomial approximation lowering.
- **Horner** Scheme



$2 \times$ degree-16



$32 \times$ degree-4

Example 1: $f(x) = \dfrac{\tanh(3x)}{\tanh(3)}$

```
--realarith-to-fixed_pt_arith="approximation-method=uniform_piecewise_poly polynomial-degree=10 coeff-storage=switch"
```

index_switch: 8 intervals (cases 0..7)

$P_0(h) = (((((((((-78 * h + 46) * h + 1454) * h + -7314) * h + -78970) * h + 682179) * h + 4256867) * h + -57922167) * h + -191490613) * h + 5510525043) * h + 5640589127)$

[...]

$P_7(h) = (((((((((83 * h + -18) * h + -206) * h + 65) * h + -477) * h + 12014) * h + -170708) * h + 1874477) * h + -15214840) * h + 81733304) * h + 30216113779)$

Towards Multi-Level Arithmetic Optimizations

EuroLLVM 2025 poster

https://hal.science/hal-05063466

3.2 Example on LLM subkernel

A Python

```python
class LlamaFfnSublayer(nn.Module):
    """Llama FFN sublayer using SwiGLU (SiLU-gated
 linear unit)."""

    def __init__(self, dim: int = 512, hidden_dim:
 int | None = None, multiple_of: int = 256):
        super().__init__()
        if hidden_dim is None:
            hidden_dim = 4 * dim
            hidden_dim = int(2 * hidden_dim / 3)
            hidden_dim = multiple_of * ((hidden_di
m + multiple_of - 1) // multiple_of)
        self.w_gate = nn.Linear(dim, hidden_dim)
        self.w_up = nn.Linear(dim, hidden_dim)
        self.w_down = nn.Linear(hidden_dim, dim)

    def forward(self, x: torch.Tensor) -> torch.Te
nsor:
        gate = F.silu(self.w_gate(x))
        up = self.w_up(x)
        return self.w_down(gate * up)
```

$$z = xW_{\text{gate}} + b_{\text{gate}}$$
$$h = \left(xW_{\text{up}} + b_{\text{up}}\right) \odot \left(\frac{z}{1+\exp(-\beta z)}\right)$$
$$y = hW_{\text{down}} + b_{\text{down}}$$

B MLIR

```
...
%8 = linalg.batch_matmul ins(%collapsed, %5
 : tensor<1x2x8xf32>, tensor<1x8x16xf32>) o
uts(%7 : tensor<1x2x16xf32>) -> tensor<1x2x
16xf32>
...
```

Matmul

C MLIR

```
...
%8 = linalg.generic {ins(%7: tensor<1x2x16x
f32>) outs(%5: tensor<1x2x16xf32>) {
  %19 = arith.negf %in : f32
  %20 = math.exp %19 : f32
  %21 = arith.addf %20, %cst_1 : f32
  %22 = arith.divf %cst_1, %21 : f32
  linalg.yield %22 : f32
} -> tensor<1x2x16xf32>

%9 = linalg.generic {ins(%8, %7: tensor<1x2
x16xf32>, tensor<1x2x16xf32>) outs(%5: tens
or<1x2x16xf32>)} {
  %19 = arith.mulf %in, %in_7 : f32
  linalg.yield %19 : f32
} -> tensor<1x2x16xf32>
...
```

D MLIR

```
...
%13 = linalg.generic ... ins(%8, %10, %transposed_4 : tenso
r<1x2x16xf32>, tensor<1x2x16xf32>, tensor<16x8xf32>) outs(%
12 : tensor<1x1x2x8xf32>) {
^bb0(%in: f32, %in_6: f32, %in_7: f32, %out: f32):
  %19 = r_arith.div(%14 : !r_arith.r_const, %18 : !r_arith.
r_expr)
  ...
  %21 = fixed_pt_arith.from_float <7, -15, signed> from %in
 : f32 {rounding = 3 : i32, saturate = true}
  %22 = fixed_pt_arith.rescale_2pow %21 : <7, -15, signed>
shift by -7
  %23 = fixed_pt_arith.extract <0, -1, signed> from %22 : <
0, -22, signed>
  %24 = fixed_pt_arith.get_int from %23 : <0, -1, signed>
  %25 = arith.index_cast %24 : i2 to index
  %26:6 = scf.index_switch %25 -> !fixed_pt_arith.fixedpt<3
, -17, signed>, !fixed_pt_arith.fixedpt<0, -17, signed>, !f
ixed_pt_arith.fixedpt<-3, -17, signed>, !fixed_pt_arith.fix
edpt<-5, -17, signed>, !fixed_pt_arith.fixedpt<-8, -17, sig
ned>, !fixed_pt_arith.fixedpt<-9, -17, signed>
  case 0 {
    %64 = fixed_pt_arith.const <66766, !fixed_pt_arith.fixe
dpt<3, -17, signed>>
...
```

`linalg-fuse-elementwise-ops` · **`promote-math-expr-to-real-arith`** · **`configure-real-arith-approx`** · **`realarith-to-fixed_pt_arith`**

3.2 Example on LLM subkernel

A Python

```python
class LlamaFfnSublayer(nn.Module):
  """Llama FFN sublayer using SwiGLU (SiLU-gated
 linear unit)."""

    def __init__(self, dim: int = 512, hidden_dim:
 int | None = None, multiple_of: int = 256):
        super().__init__()
        if hidden_dim is None:
            hidden_dim = 4 * dim
            hidden_dim = int(2 * hidden_dim / 3)
            hidden_dim = multiple_of * ((hidden_di
m + multiple_of - 1) // multiple_of)
        self.w_gate = nn.Linear(dim, hidden_dim)
        self.w_u
        self.w_d

    def forward(
nsor:
        gate = F
        up = sel
        return s
```

$$z = xW_{\text{gate}} + b_{\text{gate}}$$
$$h = (xW_{\text{up}} + b_{\text{up}}) \odot \left(\frac{z}{1+\exp(-\beta z)}\right)$$
$$y = hW_{\text{down}} + b_{\text{down}}$$

B MLIR

```
...
%8 = linalg.batch_matmul ins(%collapsed, %5
 : tensor<1x2x8xf32>, tensor<1x8x16xf32>) o
uts(%7 : tensor<1x2x16xf32>) -> tensor<1x2x
16xf32>
...
```

Matmul

C MLIR

```
...
%8 = linalg.generic {ins(%7: tensor<1x2x16x
f32>) outs(%5: tensor<1x2x16xf32>) {
  %19 = arith.negf %in : f32
...
  %19 = arith.mulf %in, %in_7 : f32
  linalg.yield %19 : f32
} -> tensor<1x2x16xf32>
...
```

D MLIR

```
...
%13 = linalg.generic ... ins(%8, %10, %transposed_4 : tenso
r<1x2x16xf32>, tensor<1x2x16xf32>, tensor<16x8xf32>) outs(%
12 : tensor<1x1x2x8xf32>) {
^bb0(%in: f32, %in_6: f32, %in_7: f32, %out: f32):
  %19 = r_arith.div(%14 : !r_arith.r_const, %18 : !r_arith.
r_expr)
  ...
  %21 = fixed_pt_arith.from_float <7, -15, signed> from %in
 : f32 {rounding = 3 : i32, saturate = true}
  %22 = fixed_pt_arith.rescale_2pow %21 : <7, -15, signed>
shift by -7
  %23 = fixed_pt_arith.extract <0, -1, signed> from %22 : <
...
  case 0 {
    %64 = fixed_pt_arith.const <66766, !fixed_pt_arith.fixe
dpt<3, -17, signed>>
...
```

High-level optimization potential:
linear algebra kernels & nonlinear functions.

linalg-fuse-elementwise-ops · promote-math-expr-to-real-arith · configure-real-arith-approx · realarith-to-fixed_pt_arith

4. Medium level *Arith* on *SCF* and *Memrefs*

4.1 Gap in MLIR → CIRCT

`--linalg-to-scf`

A SCF + MemRef + Floating Point Operations

```
...
scf.for %arg2 = %c0 to %c2 step %c1 {
  scf.for %arg3 = %c0 to %c16 step %c1 {
    scf.for %arg4 = %c0 to %c8 step %c1 {
      %3 = affine.apply #map()[%arg2, %arg4]
      %4 = memref.load %alloc_4[%3] : memref<16xf32>
      %5 = affine.apply #map1()[%arg4, %arg3]
      %6 = memref.load %alloc_5[%5] : memref<128xf32>
      %7 = affine.apply #map1()[%arg2, %arg3]
      %8 = memref.load %alloc_6[%7] : memref<32xf32>
      %9 = arith.mulf %4, %6 : f32
      %10 = arith.addf %8, %9 : f32
      memref.store %10, %alloc_6[%7] : memref<32xf32>
    }
  }
}
...
```

B Floating Point Circuit

```
...
%40 = comb.extract %1 from 31 : (i32) -> i1
%41 = comb.extract %2 from 31 : (i32) -> i1
%42 = comb.xor %40, %41 : i1
%43 = comb.concat %c0_i2, %4 : i2, i8
%44 = comb.concat %c0_i2, %22 : i2, i8
%45 = comb.concat %c1_i25, %13 : i25, i23
%46 = comb.concat %c1_i25, %31 : i25, i23
%47 = comb.mul %45, %46 : i48
%48 = comb.extract %47 from 47 : (i48) -> i1
%49 = comb.concat %c0_i9, %48 : i9, i1
%50 = comb.add %43, %44, %49, %c-127_i10 : i10
%51 = comb.concat %21, %39 : i2, i2
%52 = comb.icmp eq %51, %c4_i4 : i4
%53 = comb.icmp eq %51, %c1_i4 : i4
%54 = comb.icmp eq %51, %c0_i4 : i4
%55 = comb.or %54, %53, %52 : i1
%56 = comb.mux %55, %c0_i2, %c-1_i2 : i2
%57 = comb.icmp eq %51, %c5_i4 : i4
%58 = comb.mux %57, %c1_i2, %56 : i2
...
```

4.1 Gap in MLIR → CIRCT

--linalg-to-scf

A SCF + MemRef + Floating Point Operations

```
...
scf.for %arg2 = %c0 to %c2 step %c1 {
  scf.for %arg3 = %c0 to %c16 step %c1 {
    scf.for %arg4 = %c0 to %c8 step %c1 {
      %3 = affine.apply #map()[%arg2, %arg4]
      %4 = memref.load %alloc_4[%3] : memref<16xf32>
      %5 = affine.apply #map1()[%arg4, %arg3]
      %6 = memref.load %alloc_5[%5] : memref<128xf32>
      %7 = affine.apply #map1()[%arg2, %arg3]
      %8 = memref.load %alloc_6[%7] : memref<32xf32>
      %9 = arith.mulf %4, %6 : f32
      %10 = arith.addf %8, %9 : f32
      memref.store %10, %alloc_6[%7] : memref<32xf32>
    }
  }
}
...
```

B Floating Point Circuit

```
...
%40 = comb.extract %1 from 31 : (i32) -> i1
%41 = comb.extract %2 from 31 : (i32) -> i1
%42 = comb.xor %40, %41 : i1
%43 = comb.concat %c0_i2, %4 : i2, i8
%44 = comb.concat %c0_i2, %22 : i2, i8
%45 = comb.concat %c1_i25, %13 : i25, i23
%46 = comb.concat %c1_i25, %31 : i25, i23
%47 = comb.mul %45, %46 : i48
%48 = comb.extract %47 from 47 : (i48) -> i1
%49 = comb.concat %c0_i9, %48 : i9, i1
%50 = comb.add %43, %44, %49, %c-127_i10 : i10
%51 = comb.concat %21, %39 : i2, i2
%52 = comb.icmp eq %51, %c4_i4 : i4
%53 = comb.icmp eq %51, %c1_i4 : i4
%54 = comb.icmp eq %51, %c0_i4 : i4
%55 = comb.or %54, %53, %52 : i1
%56 = comb.mux %55, %c0_i2, %c-1_i2 : i2
%57 = comb.icmp eq %51, %c5_i4 : i4
%58 = comb.mux %57, %c1_i2, %56 : i2
...
```

How to lower towards hardware representations ? Continuing with CIRCT ?

4.1 Gap in MLIR → CIRCT

`--linalg-to-scf`

A SCF + MemRef + Floating Point Operations

```
...
scf.for %arg2 = %c0 to %c2 step %c1 {
  scf.for %arg3 = %c0 to %c16 step %c1 {
    scf.for %arg4 = %c0 to %c8 step %c1 {
      %3 = affine.apply #map()[%arg2, %arg4]
      %4 = memref.load %alloc_4[%3] : memref<16xf32>
      %5 = affine.apply #map1()[%arg4, %arg3]
      %6 = memref.load %alloc_5[%5] : memref<128xf32>
      %7 = affine.apply #map1()[%arg2, %arg3]
      %8 = memref.load %alloc_6[%7] : memref<32xf32>
      %9 = arith.mulf %4, %6 : f32
      %10 = arith.addf %8, %9 : f32
      memref.store %10, %alloc_6[%7] : memref<32xf32>
    }
  }
}
...
```

B Floating Point Circuit

```
...
%40 = comb.extract %1 from 31 : (i32) -> i1
%41 = comb.extract %2 from 31 : (i32) -> i1
%42 = comb.xor %40, %41 : i1
%43 = comb.concat %c0_i2, %4 : i2, i8
%44 = comb.concat %c0_i2, %22 : i2, i8
%45 = comb.concat %c1_i25, %13 : i25, i23
%46 = comb.concat %c1_i25, %31 : i25, i23
%47 = comb.mul %45, %46 : i48
%48 = comb.extract %47 from 47 : (i48) -> i1
%49 = comb.concat %c0_i9, %48 : i9, i1
%50 = comb.add %43, %44, %49, %c-127_i10 : i10
%51 = comb.concat %21, %39 : i2, i2
%52 = comb.icmp eq %51, %c4_i4 : i4
%53 = comb.icmp eq %51, %c1_i4 : i4
%54 = comb.icmp eq %51, %c0_i4 : i4
%55 = comb.or %54, %53, %52 : i1
%56 = comb.mux %55, %c0_i2, %c-1_i2 : i2
%57 = comb.icmp eq %51, %c5_i4 : i4
%58 = comb.mux %57, %c1_i2, %56 : i2
...
```

`--map-arith-to-comb` ?

4.1 Gap in MLIR → CIRCT

`--linalg-to-scf`

A SCF + MemRef + Floating Point Operations

```
...
scf.for %arg2 = %c0 to %c2 step %c1 {
  scf.for %arg3 = %c0 to %c16 step %c1 {
    scf.for %arg4 = %c0 to %c8 step %c1 {
      %3 = affine.apply #map()[%arg2, %arg4]
      %4 = memref.load %alloc_4[%3] : memref<16xf32>
      %5 = affine.apply #map1()[%arg4, %arg3]
      %6 = memref.load %alloc_5[%5] : memref<128xf32>
      %7 = affine.apply #map1()[%arg2, %arg3]
      %8 = memref.load %alloc_6[%7] : memref<32xf32>
      %9 = arith.mulf %4, %6 : f32
      %10 = arith.addf %8, %9 : f32
      memref.store %10, %alloc_6[%7] : memref<32xf32>
    }
  }
}
...
```

B Floating Point Circuit

```
...
%40 = comb.extract %1 from 31 : (i32) -> i1
%41 = comb.extract %2 from 31 : (i32) -> i1
%42 = comb.xor %40, %41 : i1
%43 = comb.concat %c0_i2, %4 : i2, i8
%44 = comb.concat %c0_i2, %22 : i2, i8
%45 = comb.concat %c1_i25, %13 : i25, i23
%46 = comb.concat %c1_i25, %31 : i25, i23
%47 = comb.mul %45, %46 : i48
%48 = comb.extract %47 from 47 : (i48) -> i1
%49 = comb.concat %c0_i9, %48 : i9, i1
%50 = comb.add %43, %44, %49, %c-127_i10 : i10
%51 = comb.concat %21, %39 : i2, i2
%52 = comb.icmp eq %51, %c4_i4 : i4
%53 = comb.icmp eq %51, %c1_i4 : i4
%54 = comb.icmp eq %51, %c0_i4 : i4
%55 = comb.or %54, %53, %52 : i1
%56 = comb.mux %55, %c0_i2, %c-1_i2 : i2
%57 = comb.icmp eq %51, %c5_i4 : i4
%58 = comb.mux %57, %c1_i2, %56 : i2
...
```

`--map-arith-to-comb`? No.

https://circt.llvm.org/docs/Passes/#-map-arith-to-comb: "A pass which does a simple arith to comb mapping wherever possible. This pass will not convert floating point operations."

4.2 Literature's approach

Representative approaches

- ScaleHLS / StreamTensor [1][2]
- Dynamatic [3]
- BASE2 [4]

Common limitation

Final floating-point implementation is delegated to vendor HLS or IP/core generators.

AMD/Xilinx documentation

"float and double … are synthesized with IEEE-754 standard **partial compliance**."

"complies with much of the IEEE-754 Standard … **deviations** generally provide a better trade-off of resources."

[1] Ye et al., HPCA 2022 · [2] Ye & Chen, MICRO 2025 · [3] Xu et al., FPGA 2023 · [4] Friebel et al., HEART 2023

4.2 Literature's approach

Representative approaches

- ScaleHLS / StreamTensor [1][2]
- Dynamatic [3]
- BASE2 [4]

Common limitation

Final floating-point implementation is delegated to vendor HLS or IP/core generators.

AMD/Xilinx documentation

"float and double … are synthesized with IEEE-754 standard **partial compliance**."

"complies with much of the IEEE-754 Standard … **deviations** generally provide a better trade-off of resources."

Partial compliance + hidden vendor IP => opaque arithmetic behavior and limited control.

[1] Ye et al., HPCA 2022 · [2] Ye & Chen, MICRO 2025 · [3] Xu et al., FPGA 2023 · [4] Friebel et al., HEART 2023

5. Low level *Datapaths* with *Comb*, *Seq*, and *HW*

5.1 Comb/Seq and Cost Estimation

- **CIRCT Core: Comb/Seq/HW**
- **Cost modeling**
  - AIG, MIG, High-Level Heuristics
- **Avoid commiting to hardware**
- Native AIG export: `circt-translate <synth.mlir> --export-aiger -o 90-final.aig`
- Fallback AIG export: `hls-driver/scripts/export-aig-via-yosys-abc.sh <circt-opt> <in.mlir> 90-final.aig forward`
- **PPA in seconds**
- Preparing **QoR / DSE**

F MLIR comb

```
module {
  hw.module @forward(in %arg0 : !hw.array<16xi32>, in %arg1 : !hw.array<16xi32>, in %clk :
!seq.clock, in %reset : i1, out arg1_out : !hw.array<16xi32>) {
    ...
    %111 = comb.extract %local_mem_2_rdata from 31 : (i32) -> i1
    %112 = comb.extract %local_mem_3_rdata from 31 : (i32) -> i1
    %113 = comb.xor %111, %112 : i1
    %114 = comb.extract %local_mem_2_rdata from 23 : (i32) -> i8
    %115 = comb.extract %local_mem_3_rdata from 23 : (i32) -> i8
    %118 = comb.concat %c1_i25, %116 : i25, i23
    %120 = comb.mul %118, %119 : i48
    %123 = comb.add %114, %115, %122, %c-127_i8 : i8
    ...
  }
}
```

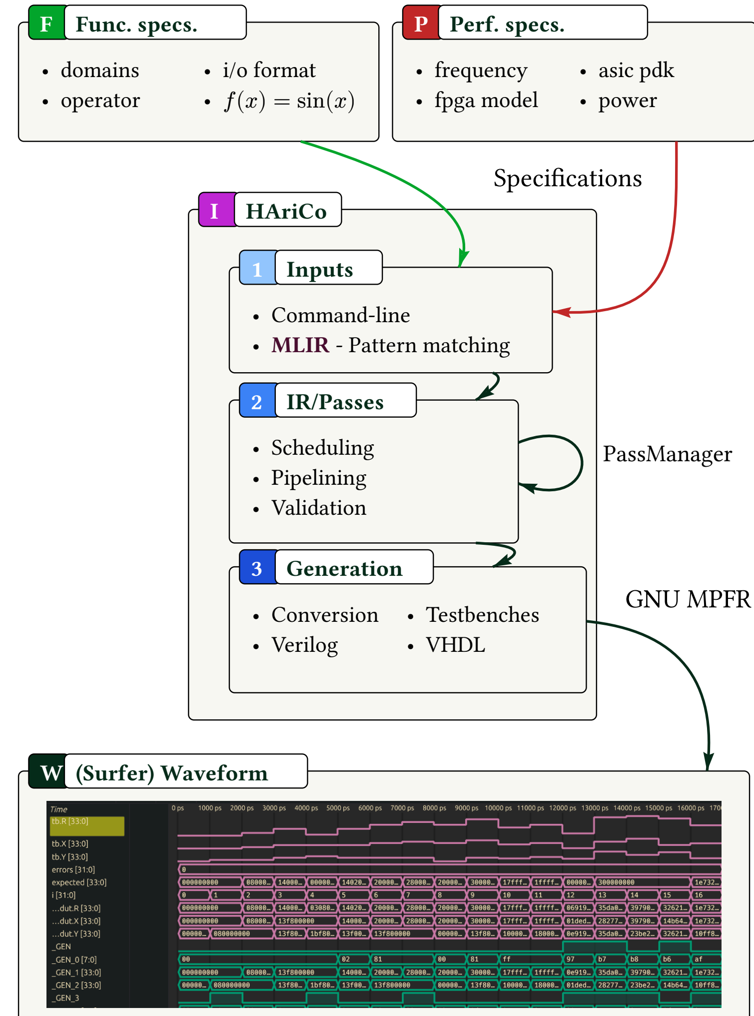
G SystemVerilog

```
module forward(
  input  [15:0][31:0] arg0,
                      arg1,
  input               clk,
                      reset,
  output [15:0][31:0] arg1_out
);
  wire [511:0] _GEN_57 = arg0;
  reg  [31:0]  arg1_mem[0:15];
  always_ff @(posedge clk) begin
    if (!reset) begin
      if (_GEN_5) arg1_mem[_GEN_1] <= _GEN_0;
      if (_GEN_9) arg1_mem[_GEN_6] <= 32'h0;
    end
  end
  ...
endmodule
```

6. HAriCo

6.1 Introducing HAriCo (Hardware Arithmetic Cores)

- IR-based **FloPoCo** implementation
  - Modernized **C++ API**
  - **Multiple outputs**
    - *VHDL, SystemVerilog, MLIR, ...*
  - Conversion & transformation **passes**
- **C API** (Python, Rust bindings)
- **Big code-base** (approx. **90** operators)
  - (will take time...)



E upstream IR (scf/arith)

```
module {
  func.func @fpmult_loop_muladd_s3fdp(
    %clk: i1 {hw.name = "clk"},
    %reset: i1 {hw.name = "reset"},
    %a: memref<2x2xf32> {hw.name = "a"},
    %b: memref<2x2xf32> {hw.name = "b"},
    %c: memref<1xf32> {hw.name = "c"})
  -> (f32 {hw.name = "r"}) {
    %c0 = arith.constant 0 : index
    %c1 = arith.constant 1 : index
    %c2 = arith.constant 2 : index
    scf.for %i = %c0 to %c2 step %c1 {
       scf.for %j = %c0 to %c2 step %c1 {
         %x = memref.load %a[%i, %j] : memref<2x2xf32>
         %y = memref.load %b[%i, %j] : memref<2x2xf32>
         %acc = memref.load %c[%c0] : memref<1xf32>
         %m = arith.mulf %x, %y : f32
         %s = arith.addf %acc, %m : f32
         memref.store %s, %c[%c0] : memref<1xf32>
       }
     }
     %r = memref.load %c[%c0] : memref<1xf32>
     func.return %r : f32
   }
}
```

6.3 Three materialization strategies

1 Per-op

- IEEE754-compliant
- 1 Operation → 1 Operator
- N Operations → N Roundings

2 Fused-graph-datapath

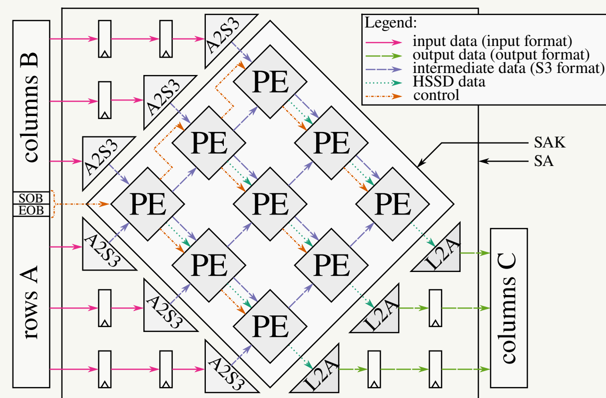
- FMA philosophy
- 1 deferred rounding
- Expression fusion

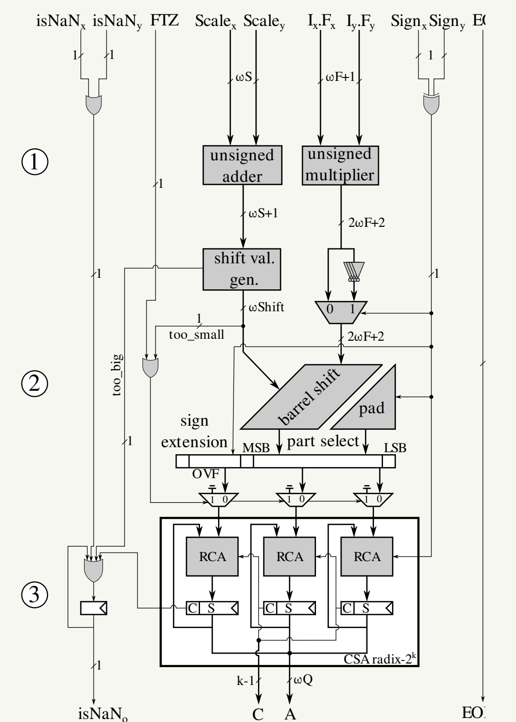
$$\frac{1}{\sqrt{x^2+y^2}}$$

$$\sin(\omega t + \varphi)$$

3 Specialized

- Squarers [1]
- Constant multipliers [2]
- Kulisch / Systolic Arrays [3]
- Target semantics





6.4 From MLIR to CIRCT: `harico-arith-to-comb`

1 Per-op

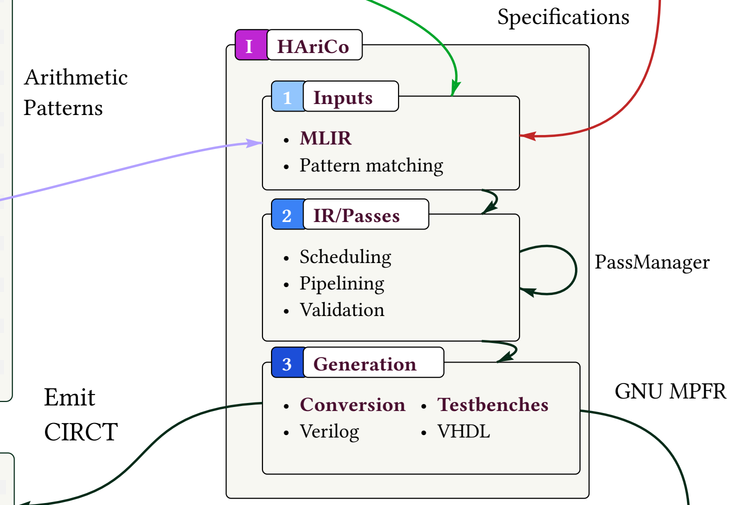
- IEEE754-compliant
- 1 Operation → 1 Operator
- N Operations → N Roundings

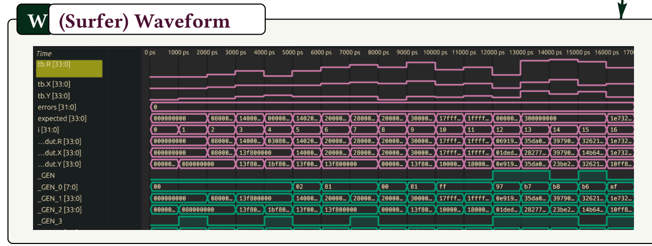
E upstream IR (scf/arith)

```
module {
  func.func @fpmult_loop_muladd_s3fdp(
    %clk: i1 {hw.name = "clk"},
    %reset: i1 {hw.name = "reset"},
    %a: memref<2x2xf32> {hw.name = "a"},
    %b: memref<2x2xf32> {hw.name = "b"},
    %c: memref<1xf32> {hw.name = "c"})
  -> (f32 {hw.name = "r"}) {
    %c0 = arith.constant 0 : index
    %c1 = arith.constant 1 : index
    %c2 = arith.constant 2 : index
    scf.for %i = %c0 to %c2 step %c1 {
      scf.for %j = %c0 to %c2 step %c1 {
        %x = memref.load %a[%i, %j] : memref<2x2xf32>
        %y = memref.load %b[%i, %j] : memref<2x2xf32>
        %acc = memref.load %c[%c0] : memref<1xf32>
        %m = arith.mulf %x, %y : f32
        %s = arith.addf %acc, %m : f32
        memref.store %s, %c[%c0] : memref<1xf32>
      }
    }
     %r = memref.load %c[%c0] : memref<1xf32>
     func.return %r : f32
   }
}
```

C CIRCT (comb/hw/seq)

```
[...]
%c0_i23 = hw.constant 0 : i23
%c0_i8 = hw.constant 0 : i8
%c0 = arith.constant 0 : index
%c1 = arith.constant 1 : index
%c2 = arith.constant 2 : index
  scf.for %arg5 = %c0 to %c2 step %c1 {
    scf.for %arg6 = %c0 to %c2 step %c1 {
      %1 = memref.load %arg2[%arg5, %arg6] : memref<2x2xi32>
      %2 = memref.load %arg3[%arg5, %arg6] : memref<2x2xi32>
      %3 = memref.load %arg4[%c0] : memref<1xi32>
      %4 = comb.extract %1 from 23 : (i32) -> i8
      %5 = comb.extract %1 from 0 : (i32) -> i23
      %6 = comb.icmp eq %4, %c0_i8 : i8
[...]
```





1. `emeraude-mlir-opt in.mlir --harico-arith-to-comb="lowering-mode=per-op target=VirtexUltraScale target-frequency=2.5e7"`

6.4 From MLIR to CIRCT: `harico-arith-to-comb`

2 Fused-graph-datapath

- FMA philosophy
- 1 deferred rounding
- Expression fusion

$$\frac{1}{\sqrt{x^2+y^2}}$$

$$\sin(\omega t + \varphi)$$

E upstream IR (scf/arith)

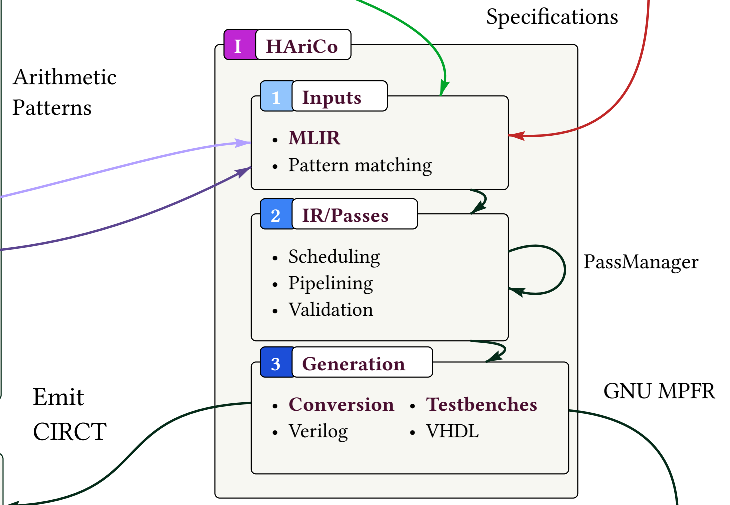
```
module {
  func.func @fpmult_loop_muladd_s3fdp(
    %clk: i1 {hw.name = "clk"},
    %reset: i1 {hw.name = "reset"},
    %a: memref<2x2xf32> {hw.name = "a"},
    %b: memref<2x2xf32> {hw.name = "b"},
    %c: memref<1xf32> {hw.name = "c"})
  -> (f32 {hw.name = "r"}) {
    %c0 = arith.constant 0 : index
    %c1 = arith.constant 1 : index
    %c2 = arith.constant 2 : index
    scf.for %i = %c0 to %c2 step %c1 {
      scf.for %j = %c0 to %c2 step %c1 {
        %x = memref.load %a[%i, %j] : memref<2x2xf32>
        %y = memref.load %b[%i, %j] : memref<2x2xf32>
        %acc = memref.load %c[%c0] : memref<1xf32>
        %m = arith.mulf %x, %y : f32
        %s = arith.addf %acc, %m : f32
        memref.store %s, %c[%c0] : memref<1xf32>
      }
    }
    %r = memref.load %c[%c0] : memref<1xf32>
    func.return %r : f32
  }
}
```

F Func. specs.

- domains
- operator
- i/o format
- $f(x) = \sin(x)$

P Perf. specs.

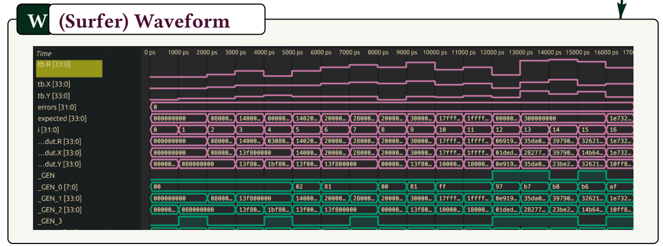
- frequency
- fpga model
- asic pdk
- power



C CIRCT (comb/hw/seq)

```
[...]
%c0_i23 = hw.constant 0 : i23
%c0_i8 = hw.constant 0 : i8
%c0 = arith.constant 0 : index
%c1 = arith.constant 1 : index
%c2 = arith.constant 2 : index
  scf.for %arg5 = %c0 to %c2 step %c1 {
    scf.for %arg6 = %c0 to %c2 step %c1 {
      %1 = memref.load %arg2[%arg5, %arg6] : memref<2x2xi32>
      %2 = memref.load %arg3[%arg5, %arg6] : memref<2x2xi32>
      %3 = memref.load %arg4[%c0] : memref<1xi32>
      %4 = comb.extract %1 from 23 : (i32) -> i8
      %5 = comb.extract %1 from 0 : (i32) -> i23
      %6 = comb.icmp eq %4, %c0_i8 : i8
[...]
```

W (Surfer) Waveform



```
2. emeraude-mlir-opt in.mlir --harico-arith-to-comb="lowering-mode=fused-graph-datapath target=sky130 target-frequency=2.5e7"
```

6.4 From MLIR to CIRCT: `harico-arith-to-comb`

3 Specialized

- Squarers
- Constant multipliers
- Kulisch / Systolic Arrays
- Target semantics

E upstream IR (scf/arith)

```
module {
  func.func @fpmult_loop_muladd_s3fdp(
    %clk: i1 {hw.name = "clk"},
    %reset: i1 {hw.name = "reset"},
    %a: memref<2x2xf32> {hw.name = "a"},
    %b: memref<2x2xf32> {hw.name = "b"},
    %c: memref<1xf32> {hw.name = "c"})
  -> (f32 {hw.name = "r"}) {
    %c0 = arith.constant 0 : index
    %c1 = arith.constant 1 : index
    %c2 = arith.constant 2 : index
    scf.for %i = %c0 to %c2 step %c1 {
      scf.for %j = %c0 to %c2 step %c1 {
        %x = memref.load %a[%i, %j] : memref<2x2xf32>
        %y = memref.load %b[%i, %j] : memref<2x2xf32>
        %acc = memref.load %c[%c0] : memref<1xf32>
        %m = arith.mulf %x, %y : f32
        %s = arith.addf %acc, %m : f32
        memref.store %s, %c[%c0] : memref<1xf32>
      }
    }
    %r = memref.load %c[%c0] : memref<1xf32>
    func.return %r : f32
  }
}
```

C CIRCT (comb/hw/seq)

```
[...]
%c0_i23 = hw.constant 0 : i23
%c0_i8 = hw.constant 0 : i8
%c0 = arith.constant 0 : index
%c1 = arith.constant 1 : index
%c2 = arith.constant 2 : index
  scf.for %arg5 = %c0 to %c2 step %c1 {
    scf.for %arg6 = %c0 to %c2 step %c1 {
      %1 = memref.load %arg2[%arg5, %arg6] : memref<2x2xi32>
      %2 = memref.load %arg3[%arg5, %arg6] : memref<2x2xi32>
      %3 = memref.load %arg4[%c0] : memref<1xi32>
      %4 = comb.extract %1 from 23 : (i32) -> i8
      %5 = comb.extract %1 from 0 : (i32) -> i23
      %6 = comb.icmp eq %4, %c0_i8 : i8
[...]
```


3. `emeraude-mlir-opt in.mlir --harico-arith-to-comb="lowering-mode=specialized specializations=enable=s3fdp,s3fdp.chunk_size=32,s3fdp.ovf=10,s3fdp.msb=10,s3fdp.lsb=-20 target=tsmc7 target-frequency=2.5e7"`

6.5 HAriCo C++ API example

C++ API | cpp

```cpp
#include <HAriCo/Context.hpp>
#include <HAriCo/Operators/FPAddSinglePath.hpp>
using namespace HAriCo;

auto ctx = Context::new_default();
ctx.options["target"] = Targets::VirtexUltraScalePlus;
ctx.options["frequency"] = 500 // MHz;

// Build operator:
auto op = FPAdd();
auto options = op.interface(ctx);
options["wE"] = 8;
options["wF"] = 23;
Module mod = op.build(ctx, options);

// Before pass:
VerilogCodeGenerator gen;
gen.build(mod).fwrite("fpadd.sv");

PassManager pm;
pm.add<TimingAnalysisPass>();
pm.add<SchedulePass>();
pm.add<PipelineizePass>();
pm.run(mod);
gen.build(mod).fwrite("fpadd_pipeline.sv");
```

Global options

IR build

HDL generation

Analysis & transformation passes

6.6 HAriCo Python bindings example

HAriCo.py Python

```python
import harico as hrc

def main():
    ctx = hrc.Context()
    ctx.set_target("VirtexUltrascalePlus", frequency_mhz=500)

    op = hrc.Operator("FPAdd")
    mod = op.build_ir(
        ctx,
        wE=8,
        wF=23,
        sub=False,
        dualPath=False,
        onlyPositiveIO=False,
    )

    # Before pass:
    mod.emit_vhdl("fpadd_before_pipelineize.vhd")
    mod.emit_verilog("fpadd_before_pipelineize.v")

    # Analysis passes first:
    pm_analysis = hrc.PassManager(["timing", "schedule"])
    pm_analysis.run(mod, ctx)

    # Transform pass:
    pm_pipelineize = hrc.PassManager(["pipelineize"])
    pm_pipelineize.run(mod, ctx)
```

Python bindings (import line)

IR build (Operator / build_ir block)

HDL generation (emit_vhdl / emit_verilog block)

Analysis & transformation passes (PassManager blocks)

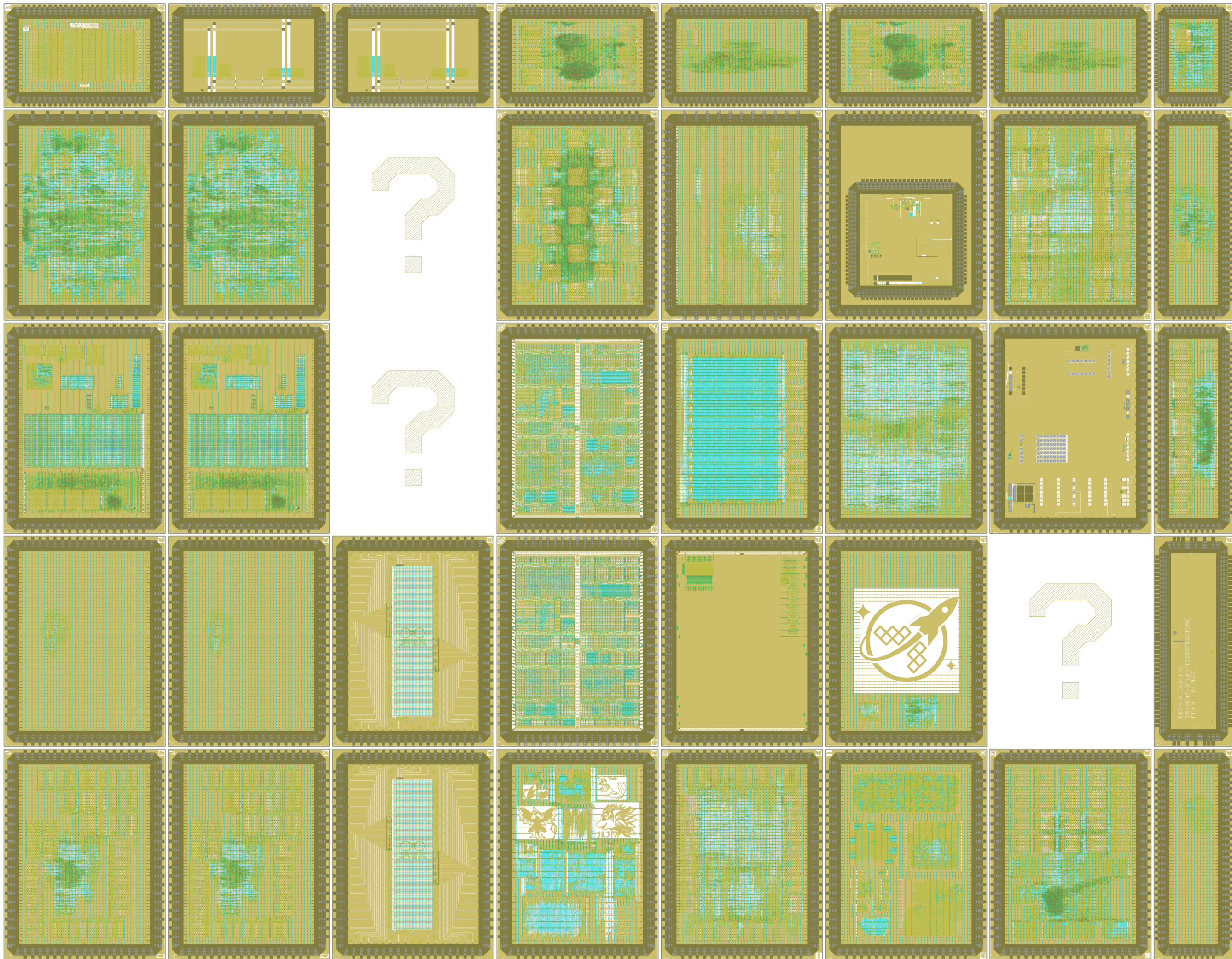
7. Results (DSP)

7.1 Wafer.space run

Figure 1: MPW + TT submission (TTg0p2) on Wafer.space — Utilisation 47.106% · Wire length 377542 um

7.2 GDS 3D view

7.3 Reticle view (wafer.space)



8. Results (LLM & HAriCo lowering strategies)

8.1 ASIC: GDS routing congestion

ASIC SKY130HD Llama

Metric	Value
Area	$1.7233 mm^2$
Die Dimensions	1.312 x 1.312 mm
Power Breakdown	
Combinational	0.117 W
Sequential	0.044 W
Clock	0.028 W
Total Power	0.189 W

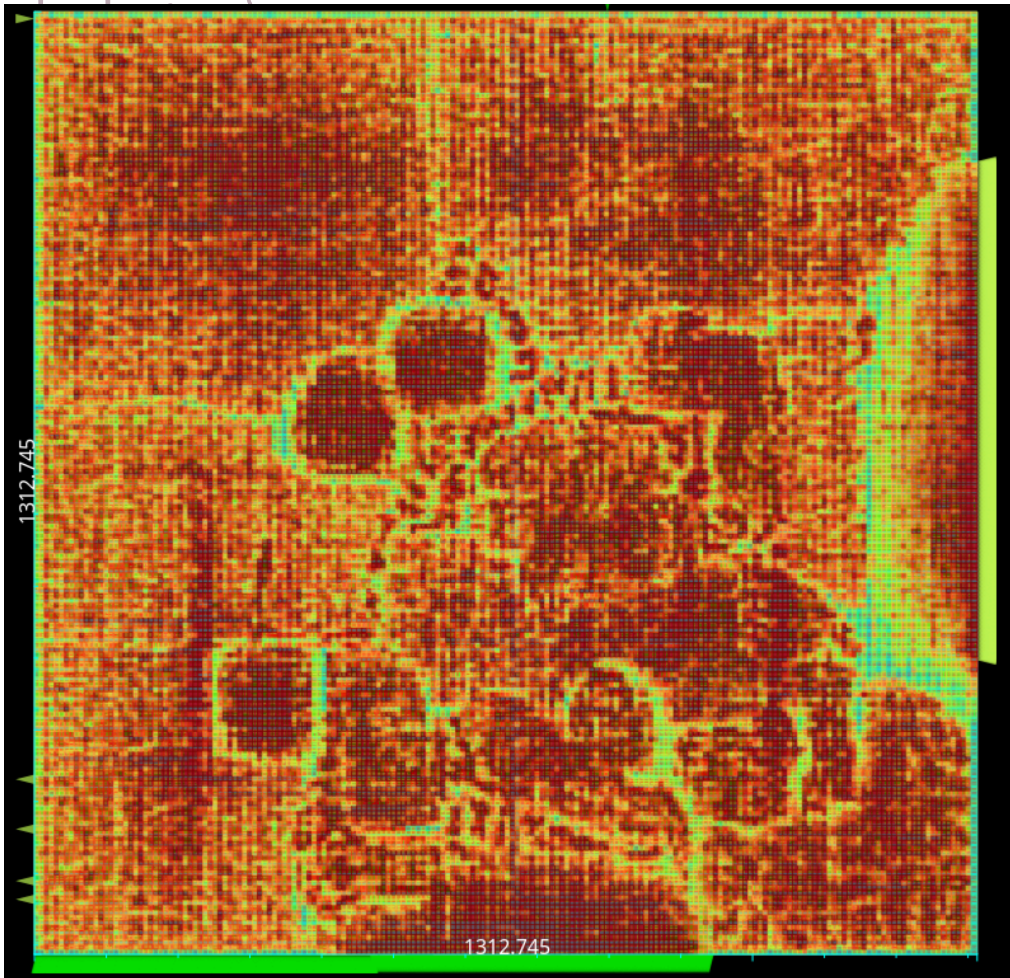
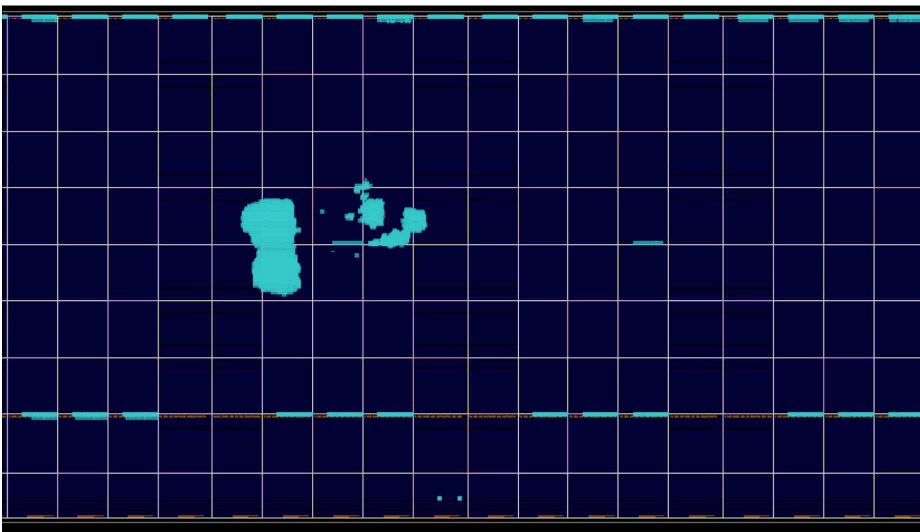


Figure 2: Llama 1.5, mm^2 routing congestion heatmap

8.2 FPGA: Ressources usage

FPGA VU19P Llama

Metric	Value
LUTs	26,928
Flip-Flops	12,221
DSP Blocks	17



Llama FPGA vu19p P&R

8.3 Llama FFN Sublayer

Llama FFN Sublayer IR | IR

```
module {
  func.func @forward(%arg0: tensor<1x2x8xf32>)
-> tensor<1x2x8xf32> {
    %7 = linalg.batch_matmul ... -> tensor<1x2x
16xf32>
    %8 = linalg.generic ... ins(%7 : tensor<1x2
x16xf32>) outs(%5 : tensor<1x2x16xf32>) {
    ^bb0(%in: f32, %out: f32):
      %neg = arith.negf %in : f32
      %exp = math.exp %neg : f32
      %gate = arith.divf %cst_1, %den : f32
      %silu = arith.mulf %in, %gate : f32
      linalg.yield %silu : f32
    } -> tensor<1x2x16xf32>
    return %18 : tensor<1x2x8xf32>
  }
}
```

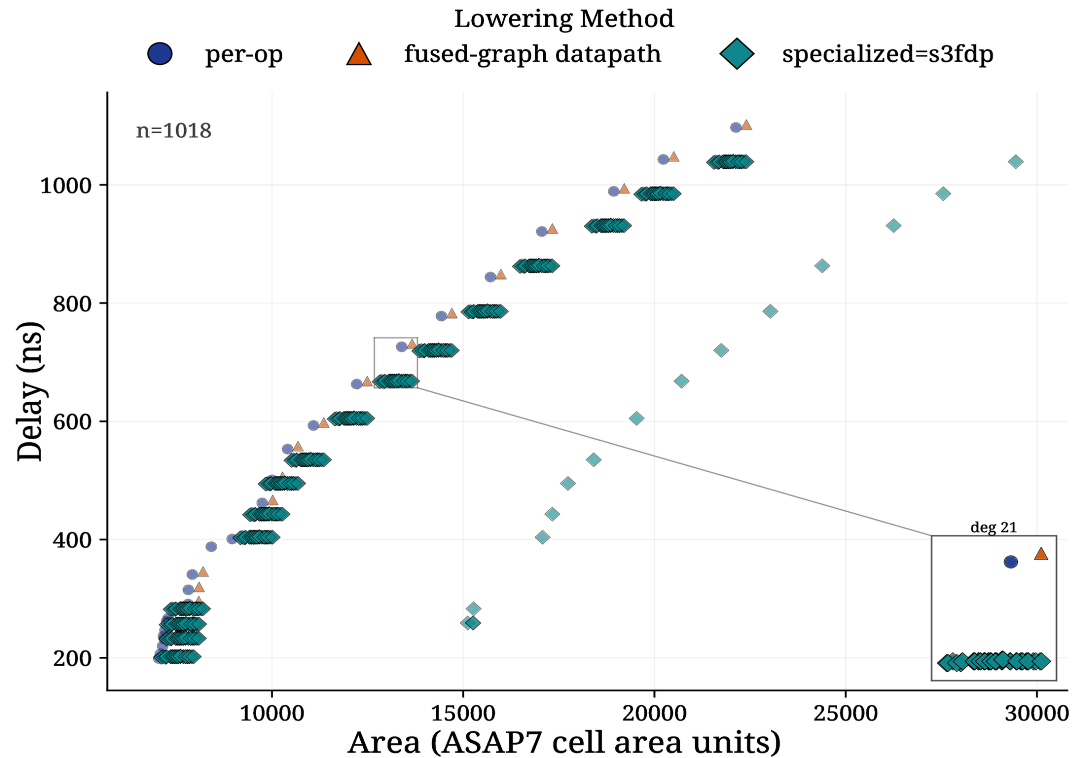


Figure 3: Llama FFN Sublayer. Area vs. Delay

8.4 MatMul + SiLU

MatMul SiLU IR | IR

```
#map = affine_map<(d0, d1) -> (d0, d1)>
module {
  func.func @forward(%arg0: tensor<2x4xf32>) ->
 tensor<2x4xf32> {
    %mm = linalg.matmul ... -> tensor<2x4xf32>
    %sig = linalg.generic ... ins(%mm : tensor<
2x4xf32>) outs(%empty : tensor<2x4xf32>) {
    ^bb0(%in: f32, %acc: f32):
      %neg = arith.negf %in : f32
      %exp = math.exp %neg : f32
      %gate = arith.divf %c1, %den : f32
      linalg.yield %gate : f32
    } -> tensor<2x4xf32>
    %out = linalg.generic ... ins(%sig, %mm : t
ensor<2x4xf32>, tensor<2x4xf32>) outs(%empty :
tensor<2x4xf32>)
    return %out : tensor<2x4xf32>
  }
}
```

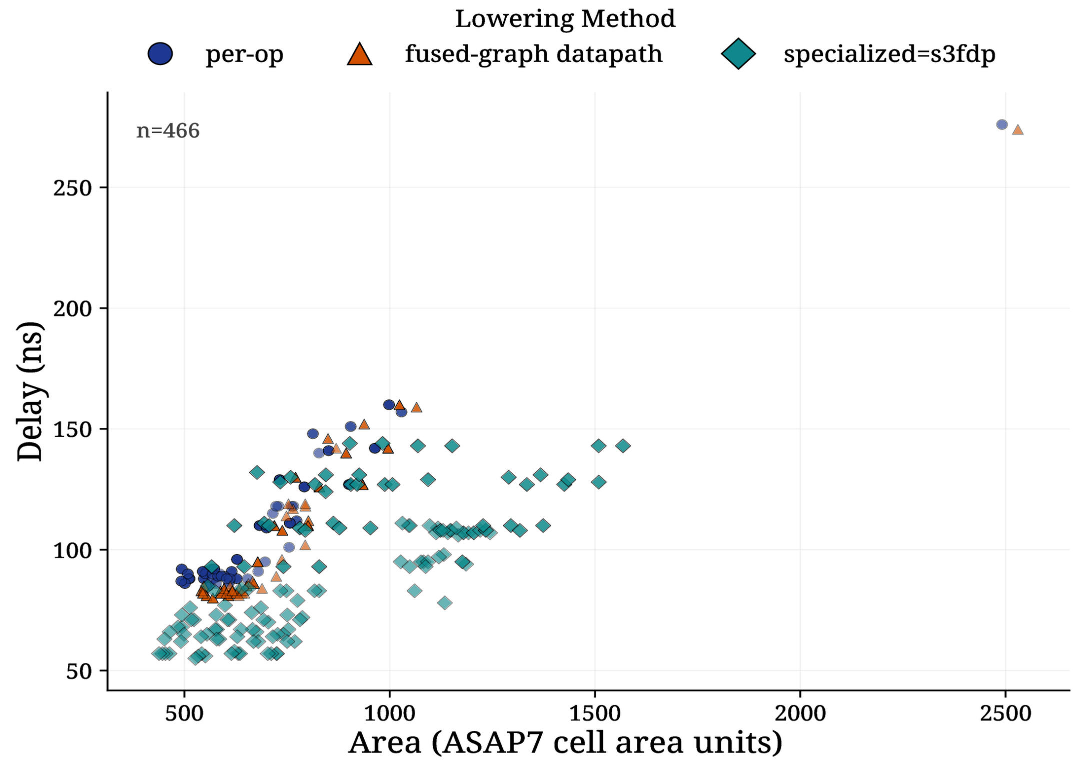


Figure 4: MatMul + SiLU. Area vs. Delay

8.5 MatMul + Sigmoid

MatMul Sigmoid IR | IR

```
#map = affine_map<(d0, d1) -> (d0, d1)>
module {
  func.func @forward(%arg0: tensor<2x4xf32>) ->
 tensor<2x4xf32> {
    %mm = linalg.matmul ... -> tensor<2x4xf32>
    %out = linalg.generic ... ins(%mm : tensor<
2x4xf32>) outs(%empty : tensor<2x4xf32>) {
    ^bb0(%in: f32, %acc: f32):
      %neg = arith.negf %in : f32
      %exp = math.exp %neg : f32
      %sig = arith.divf %c1, %den : f32
      linalg.yield %sig : f32
    } -> tensor<2x4xf32>
    return %out : tensor<2x4xf32>
  }
}
```

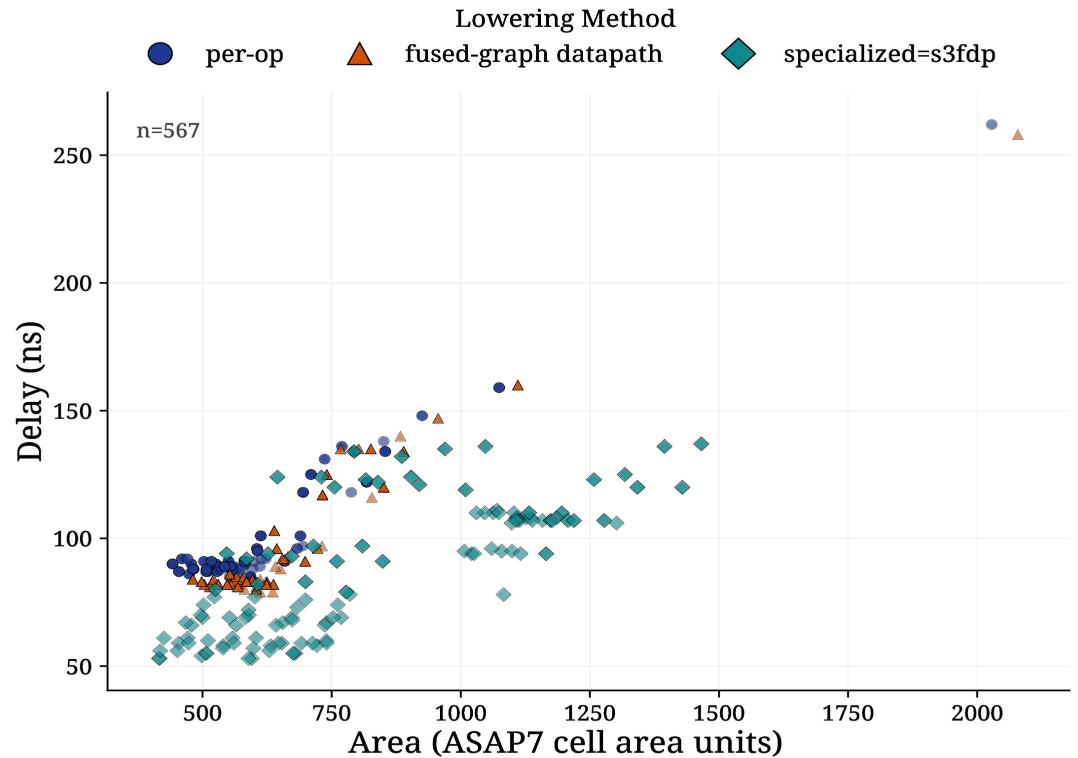


Figure 5: MatMul + Sigmoid. Area vs. Delay

8.6 Attention Softmax

Attention Softmax Exponential IR | IR

```
#map = affine_map<(d0, d1) -> (d0, d1)>
module {
  func.func @forward(%query: tensor<2x4xf32>) -
> tensor<2x2xf32> {
    %score = linalg.matmul ... -> tensor<2x2xf3
2>
    %out = linalg.generic ... ins(%score : tens
or<2x2xf32>) outs(%out_empty : tensor<2x2xf32>)
 {
    ^bb0(%in: f32, %acc: f32):
      %res = math.exp %in : f32
      linalg.yield %res : f32
    } -> tensor<2x2xf32>
    return %out : tensor<2x2xf32>
  }
}
```

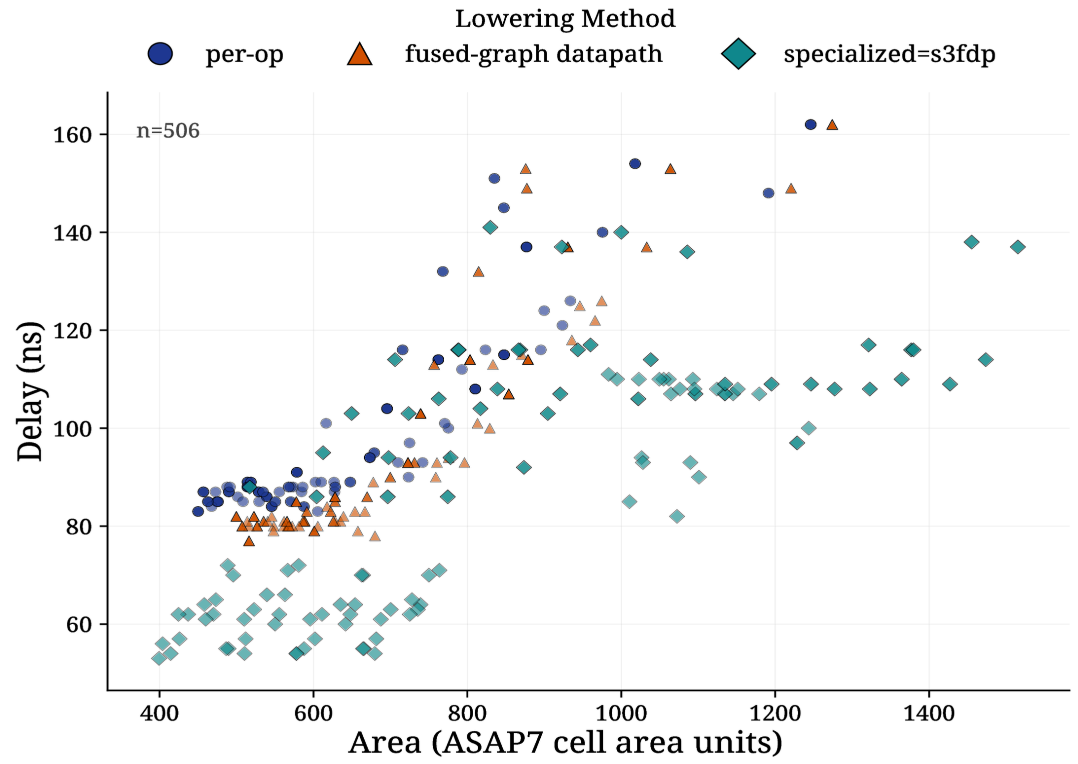


Figure 6: Attention Softmax Exponential. Area vs. Delay

8.7 MatMul + Sigmoid Heavy-Tail

MatMul Sigmoid Heavy-Tail IR | IR

```
#map = affine_map<(d0, d1) -> (d0, d1)>
module {
  func.func @forward(%arg0: tensor<2x8xf32>) ->
 tensor<2x8xf32> {
     %mm = linalg.matmul ... -> tensor<2x8xf32>
     %out = linalg.generic ... ins(%mm : tensor<
2x8xf32>) outs(%empty : tensor<2x8xf32>) {
     ^bb0(%in: f32, %acc: f32):
       %neg = arith.negf %in : f32
       %exp = math.exp %neg : f32
       %res = arith.divf %c1, %den : f32
       linalg.yield %res : f32
     } -> tensor<2x8xf32>
     return %out : tensor<2x8xf32>
  }
}
```

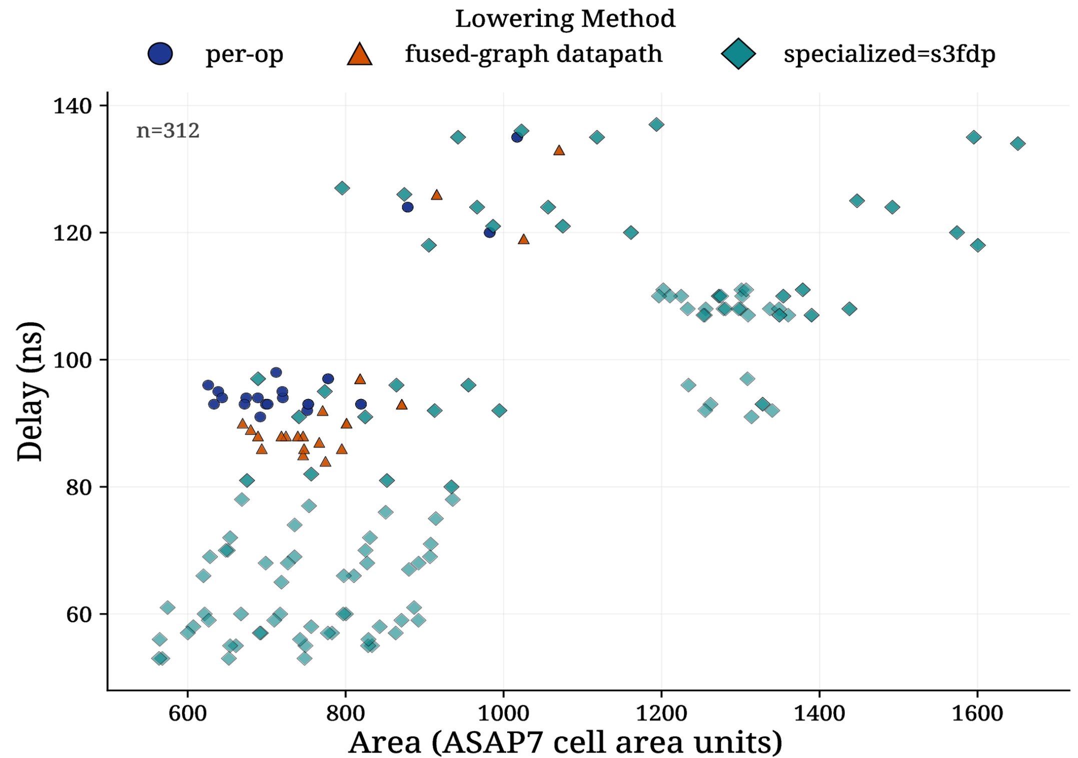


Figure 7: MatMul + Sigmoid Heavy-Tail. Area vs. Delay

8.8 GEMV Accumulation

GEMV Accumulation IR | IR

```
module {
  func.func @forward(%arg0: tensor<1x4xf32>) ->
 tensor<1x4xf32> {
    %w = arith.constant dense<...> : tensor<4x4
xf32>
    %init = linalg.fill ins(%c0 : f32) outs(%em
pty : tensor<1x4xf32>) -> tensor<1x4xf32>
    %out = linalg.matmul ins(%arg0, %w : tensor
<1x4xf32>, tensor<4x4xf32>) outs(%init : tensor
<1x4xf32>)
          -> tensor<1x4xf32>
    return %out : tensor<1x4xf32>
  }
}
```

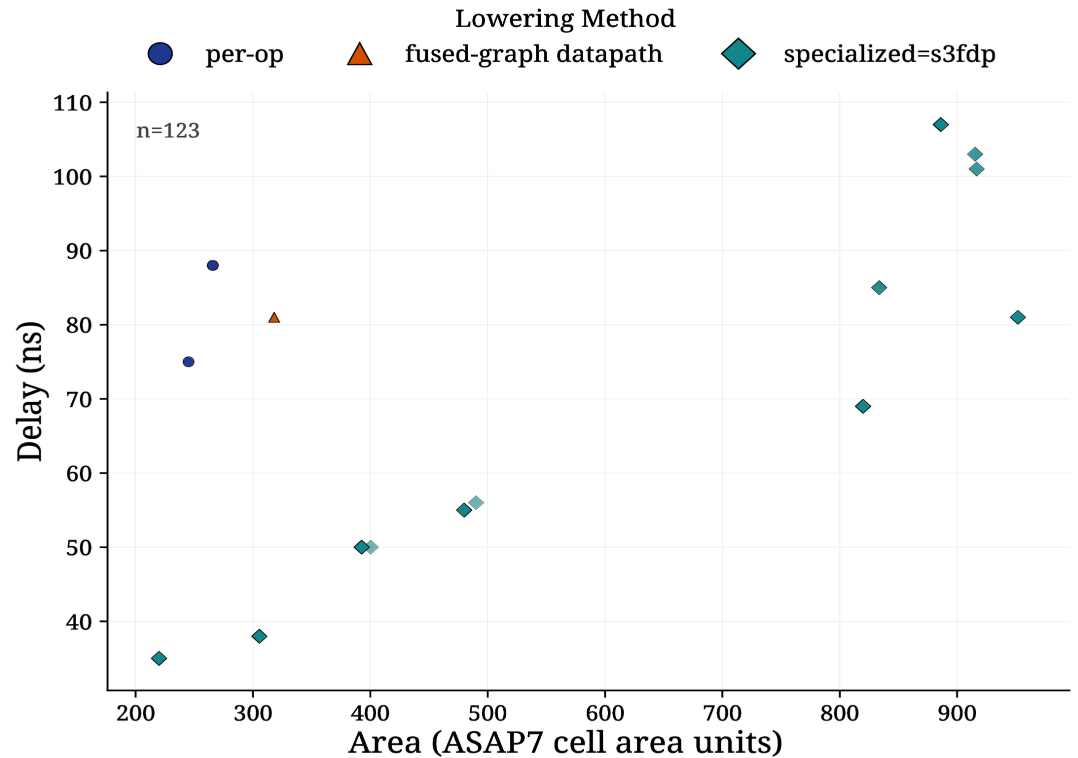


Figure 8: GEMV Accumulation. Area vs. Delay

8.9 Attention Score

Attention Score IR | IR

```
module {
  func.func @forward(%query: tensor<2x4xf32>) -> tensor<2x2xf32> {
    %kt = arith.constant dense<...> : tensor<4x2xf32>
    %init = linalg.fill ins(%c0 : f32) outs(%empty : tensor<2x2xf32>) -> tensor<2x2xf32>
    %score = linalg.matmul ins(%query, %kt : tensor<2x4xf32>, tensor<4x2xf32>) outs(%init : tensor<2x2xf32>)
      -> tensor<2x2xf32>
    return %score : tensor<2x2xf32>
  }
}
```

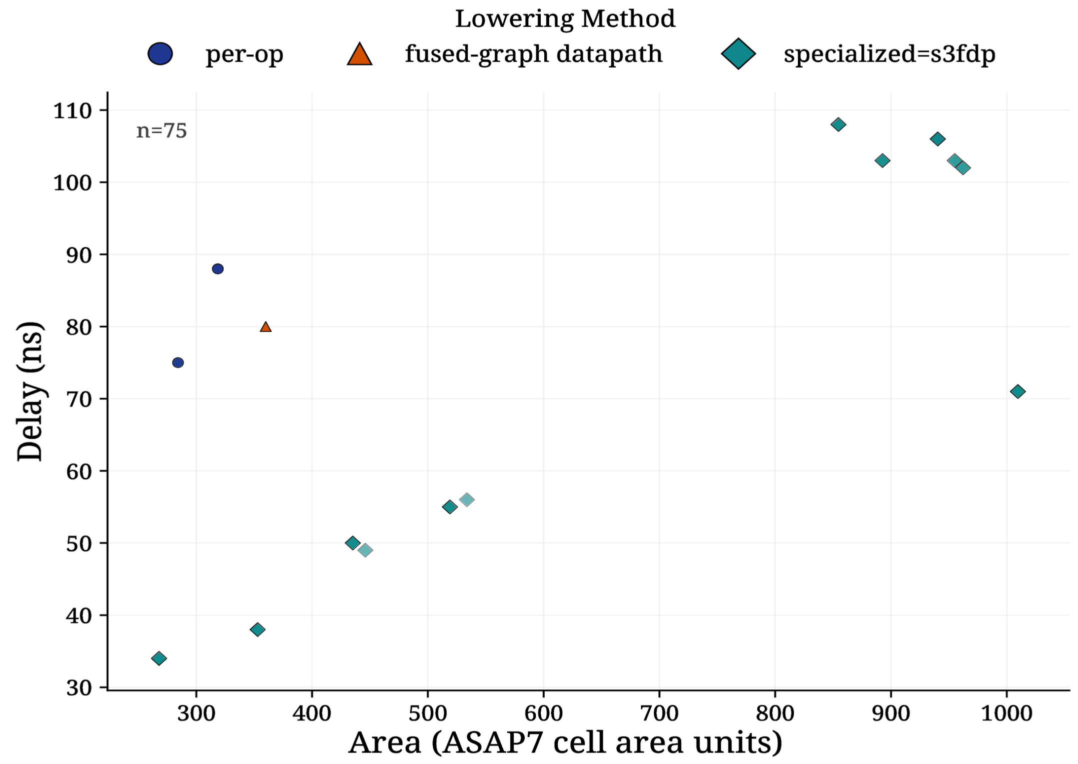


Figure 9: Attention Score. Area vs. Delay

8.10 MatMul Accumulation

MatMul Accumulation IR | IR

```
module {
  func.func @forward(%arg0: tensor<2x4xf32>) ->
 tensor<2x4xf32> {
    %w = arith.constant dense<...> : tensor<4x4
xf32>
    %init = linalg.fill ins(%c0 : f32) outs(%em
pty : tensor<2x4xf32>) -> tensor<2x4xf32>
    %mm = linalg.matmul ins(%arg0, %w : tensor<
2x4xf32>, tensor<4x4xf32>) outs(%init : tensor<
2x4xf32>)
       -> tensor<2x4xf32>
    return %mm : tensor<2x4xf32>
  }
}
```

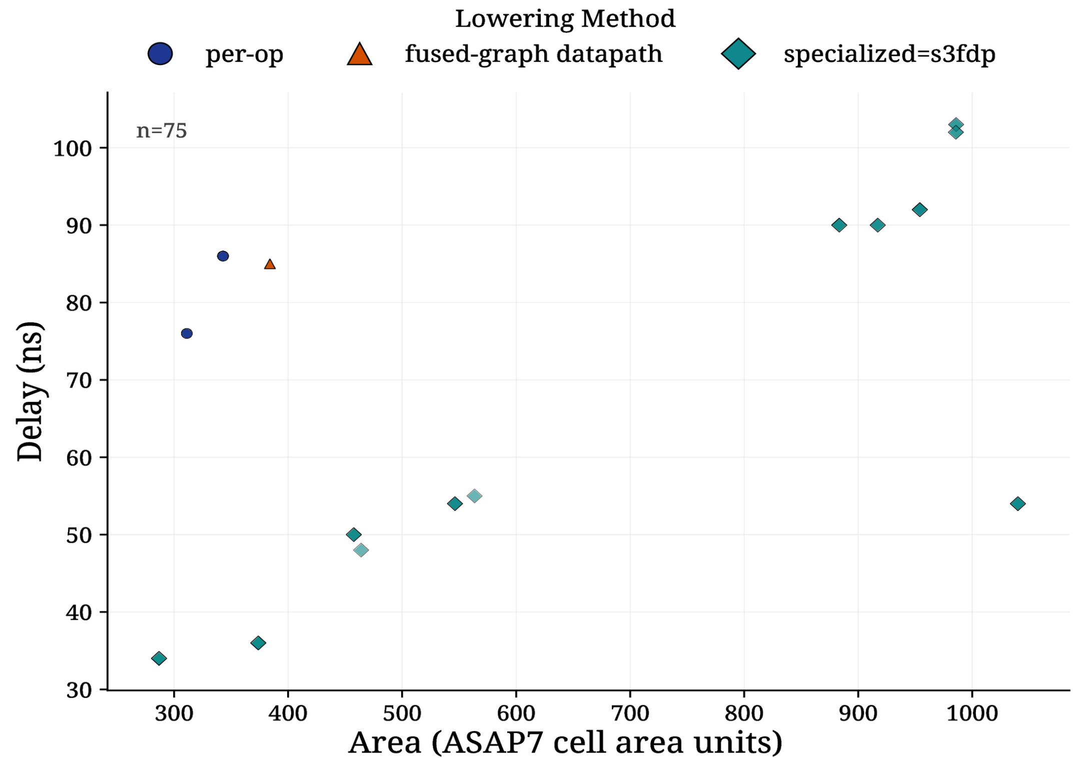


Figure 10: MatMul Accumulation. Area vs. Delay

8.11 Polybench GESUMMV-Like

Polybench GESUMMV-Like IR | IR

```
#map = affine_map<(d0, d1) -> (d0, d1)>
module {
  func.func @forward(%arg0: tensor<1x4xf32>) ->
 tensor<1x4xf32> {
    %mm = linalg.matmul ... -> tensor<1x4xf32>
    %out = linalg.generic ... ins(%mm, %arg0 :
tensor<1x4xf32>, tensor<1x4xf32>) outs(%empty_o
ut : tensor<1x4xf32>) {
    ^bb0(%mmv: f32, %in: f32, %acc: f32):
      %bias = arith.mulf %beta, %in : f32
      %res = arith.addf %mmv, %bias : f32
      linalg.yield %res : f32
    } -> tensor<1x4xf32>
    return %out : tensor<1x4xf32>
  }
}
```

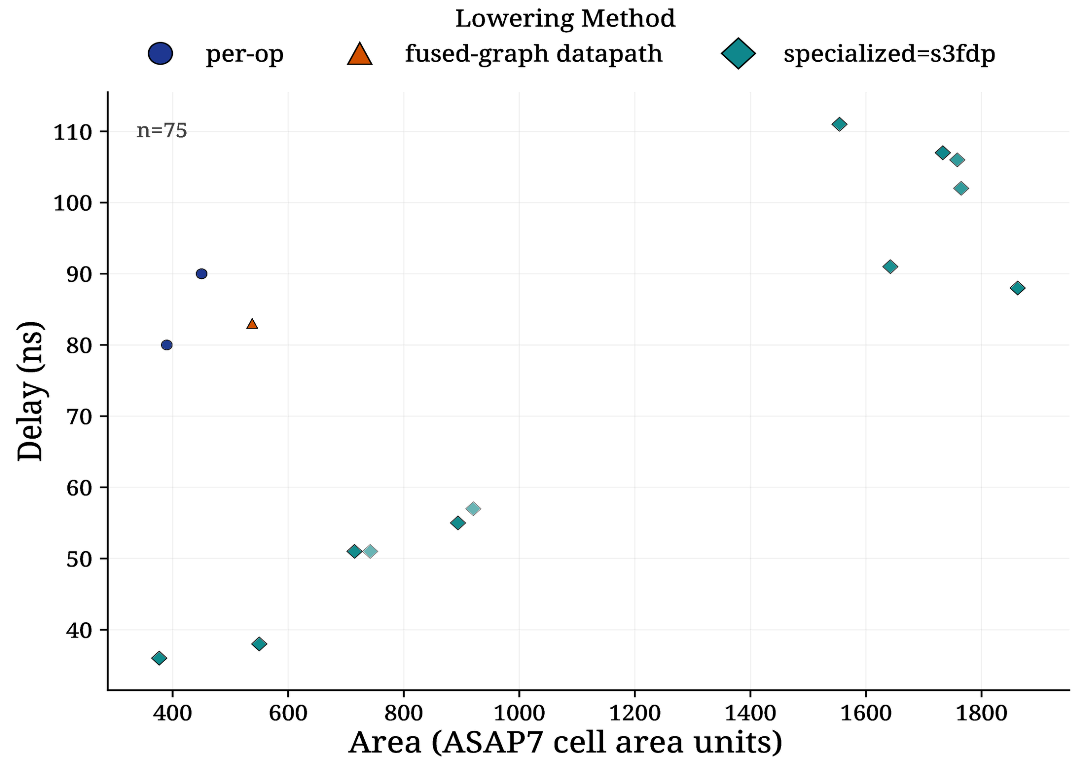


Figure 11: Polybench GESUMMV-Like. Area vs. Delay

8.12 DSP FIR Accumulation

DSP FIR Accumulation IR | IR

```
module {
  func.func @forward(%arg0: tensor<1x8xf32>) ->
 tensor<1x8xf32> {
    %w = arith.constant dense<...> : tensor<8x8
xf32>
    %init = linalg.fill ins(%c0 : f32) outs(%em
pty : tensor<1x8xf32>) -> tensor<1x8xf32>
    %mm = linalg.matmul ins(%arg0, %w : tensor<
1x8xf32>, tensor<8x8xf32>) outs(%init : tensor<
1x8xf32>)
       -> tensor<1x8xf32>
    return %mm : tensor<1x8xf32>
  }
}
```

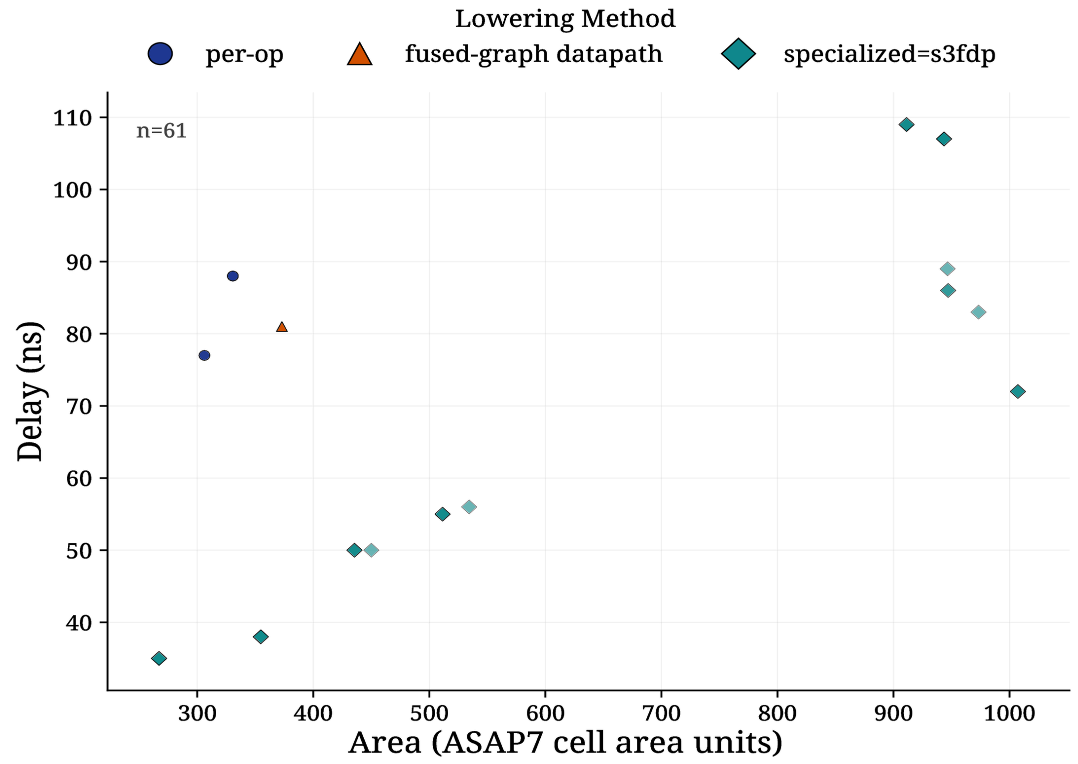


Figure 12: DSP FIR Accumulation. Area vs. Delay

8.13 Polybench SYRK-Like

Polybench SYRK-Like IR | IR

```
module {
  func.func @forward(%arg0: tensor<1x8xf32>) ->
 tensor<1x8xf32> {
    %w = arith.constant dense<...> : tensor<8x8
xf32>
    %init = linalg.fill ins(%c0 : f32) outs(%em
pty : tensor<1x8xf32>) -> tensor<1x8xf32>
    %mm = linalg.matmul ins(%arg0, %w : tensor<
1x8xf32>, tensor<8x8xf32>) outs(%init : tensor<
1x8xf32>)
       -> tensor<1x8xf32>
    return %mm : tensor<1x8xf32>
  }
}
```

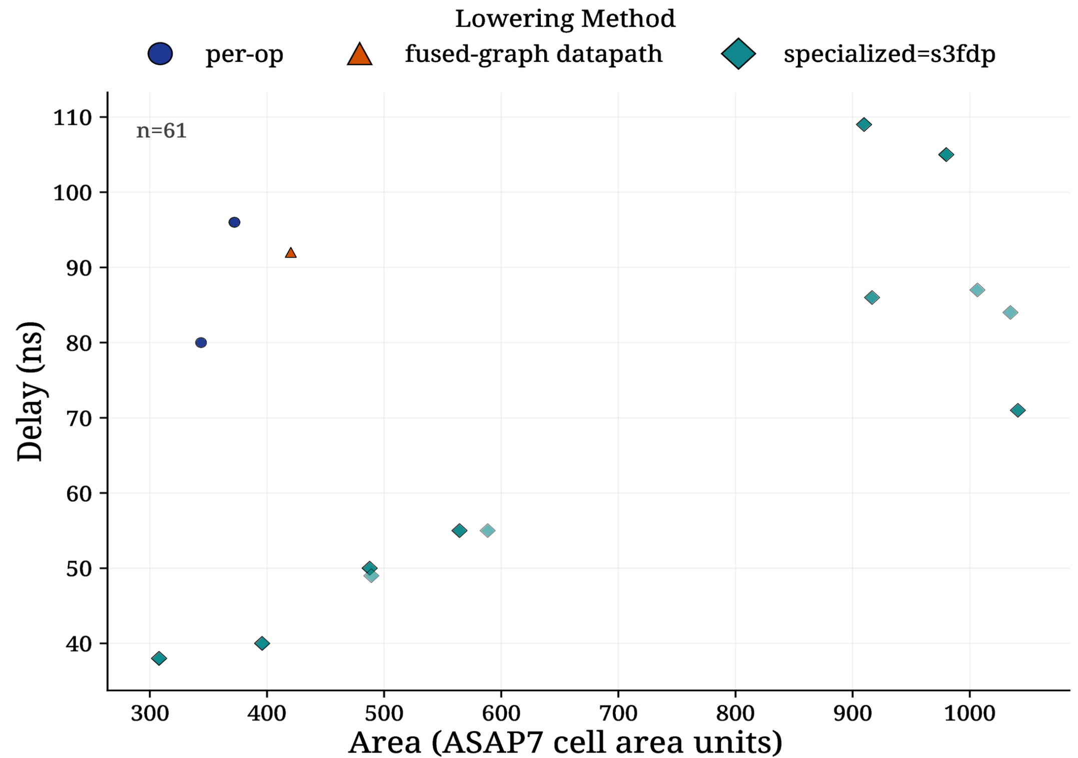


Figure 13: Polybench SYRK-Like. Area vs. Delay

9. Closing Slides

9.1 CIRCT / LLVM commits

llvm / **circt**

Code Issues 849 Pull requests 193 Actions Security Insights

Filters is:pr author:Bynaryman is:closed | Labels 53 | Milestones 3 | New pull request

Clear current search query, filters, and sorts

0 Open ✓ **2 Closed**

Author Label Projects Milestones Reviews Assignee Sort

[Transform] Add convert-index-to-uint transform to normalize index compares before comb mapping ✓ 15

#9263 by Bynaryman was merged on Dec 2, 2025 • Approved

[SwitchToIf] support multi-result scf.index_switch + add regression test ✓ 7

#9245 by Bynaryman was merged on Nov 27, 2025 • Approved

Internal Emeraude-MLIR transforms pushed upstream.

9.2 Concluding remarks (Future Work)

- HL: More polynomial approximations schemes
- ML: More pattern matching in specialized
- LL: Scheduling transform at *Comb* level, Bitheaps

9.2 Concluding remarks (Future Work)

- HL: More polynomial approximations schemes
- ML: More pattern matching in specialized
- LL: Scheduling transform at *Comb* level, Bitheaps
- The DSE engine [4], [5][1][2]

[1]Hanchen Ye et al., “ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation,” HPCA 2022.

[2]Hanchen Ye and Deming Chen, “StreamTensor: Make Tensors Stream in Dataflow Accelerators for LLMs,” MICRO 2025.

9.2 Concluding remarks (Future Work)

- HL: More polynomial approximations schemes
- ML: More pattern matching in specialized
- LL: Scheduling transform at *Comb* level, Bitheaps
- The DSE engine [4], [5][1][2]
- Open the Repository at some point

[1] Hanchen Ye et al., “ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation,” HPCA 2022.

[2] Hanchen Ye and Deming Chen, “StreamTensor: Make Tensors Stream in Dataflow Accelerators for LLMs,” MICRO 2025.

9.2 Concluding remarks (Future Work)

- HL: More polynomial approximations schemes
- ML: More pattern matching in specialized
- LL: Scheduling transform at *Comb* level, Bitheaps
- The DSE engine [4], [5][1][2]
- Open the Repository at some point

Overall…

- Perfect for collaborations
  - RSCM, Optimization DSE, arithmetic, audio…
  - Interns armies?
  - External (Datapath dialect, CIRCT, EDA community, etc.)

---

[1] Hanchen Ye et al., “ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation,” HPCA 2022.

[2] Hanchen Ye and Deming Chen, “StreamTensor: Make Tensors Stream in Dataflow Accelerators for LLMs,” MICRO 2025.

9.3 Demo time :)

I will just execute a command while speaking

9.4 Thank you

Thank you

Q&A

Bibliography

[1] F. de Dinechin and M. Kumm, *Application-Specific Arithmetic*. Springer.

[2] B. Barbe, L. Ledoux, A. Volkova, and F. de Dinechin, "Reconfigurable constant multipliers: Hardware models, optimization algorithm and applications," *Microprocessors and Microsystems*, p. 105270, 2026, doi: https://doi.org/10.1016/j.micpro.2026.105270.

[3] L. Ledoux and M. Casas, "An Open-Source Framework for Efficient Numerically-Tailored Computations," in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, 2023, pp. 19–26. doi: 10.1109/FPL60245.2023.00011.

[4] H. Ye *et al.*, "ScaleHLS: A New Scalable High-Level Synthesis Framework on Multi-Level Intermediate Representation," in *2022 IEEE International*

Symposium on High-Performance Computer Architecture (HPCA), 2022, pp. 741–755. doi: 10.1109/HPCA53966.2022.00060.

[5] H. Ye and D. Chen, “StreamTensor: Make Tensors Stream in Dataflow Accelerators for LLMs,” in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, in MICRO '25. Association for Computing Machinery, 2025, pp. 201–216. doi: 10.1145/3725843.3762817.

10. Appendix

10.1 GDS cell usage

Cell usage by Category

Category	Cells	Count
Fill	fillcap fill	10280
NAND	nand2 nand4 nand3	2627
Combo Logic	aoi21 oai21 oai22 oai31 aoi211 oai211 aoi22 oai32 oai221 aoi221 oai33	1645
Buffer	clkbuf dlyb buf dlya	1258
Flip Flops	dffq	1179
NOR	nor2 nor4 nor3 xnor2 xnor3	1024
OR	or2 or3 or4 xor2 xor3	1020
Clock	clkinv	367
AND	and2 and3 and4	323
Multiplexer	mux2	316
Inverter	inv	55

All in the web as Github Actions.. Impressive !

10.2 GDS precheck results

Tiny Tapeout Precheck Results

Check	Result
Magic DRC	✅
KLayout pin label overlapping drawing	✅
KLayout zero area	✅
KLayout Checks	✅
Pin check	✅
Boundary check	✅
Power pin check	✅
Layer check	✅
Cell name check	✅
Analog pin check	✅

Many transforms from one Faust line of code passing all checks :)