E-Graphs Talk: 12th Nov

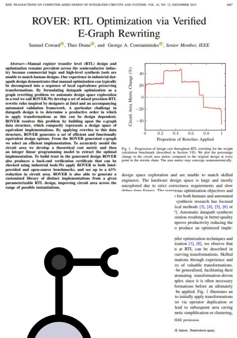
Synthesizing Faster Circuits using Redundant Representations

Sam Coward - Research Fellow, University College London

Collaborators - Cambridge (Grosser Lab), SiFive

Introduction

IMPERIAL



IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 43, NO. 12, DECEMBER 2024 4687

ROVER: RTL Optimization via Verified E-Graph Rewriting

Samuel Coward, Theo Drane, and George A. Constantinides, *Senior Member, IEEE*

***Abstract*—Manual register transfer level (RTL) design and optimization remains prevalent across the semiconductor industry because commercial logic and high-level synthesis tools are unable to match human designs. Our experience in industrial datapath design demonstrates that manual optimization can typically be decomposed into a sequence of local equivalence preserving transformations. By formulating datapath optimization as a graph rewriting problem we automate design space exploration in a tool we call ROVER.We develop a set of mixed precision RTL rewrite rules inspired by designers at Intel and an accompanying automated validation framework. A particular challenge in datapath design is to determine a productive order in which to apply transformations as this can be design dependent. ROVER resolves this problem by building upon the e-graph data structure, which compactly represents a design space of equivalent implementations. By applying rewrites to this data structure, ROVER generates a set of efficient and functionally equivalent design options. From the ROVER generated e-graph we select an efficient implementation. To accurately model the circuit area we develop a theoretical cost metric and then an integer linear programming model to extract the optimal implementation. To build trust in the generated design ROVER also produces a back-end verification certificate that can be checked using industrial tools.We apply ROVER to both Intel-provided and open-source benchmarks, and see up to a 63% reduction in circuit area. ROVER is also able to generate a customized library of distinct implementations from a given parameterizable RTL design, improving circuit area across the range of possible instantiations.**

Circuit Area Metric Change (%)
40
20
0
−20
−40
0
0.2
0.4
0.6
0.8
1
Proportion of Rewrites Applied

Fig. 1. Progression of design cost throughout RTL rewriting for the weight calculation benchmark (described in Section VII). We plot the percentage change in the circuit area metric compared to the original design at every point in the rewrite chain. The area metric may converge nonmonotonically.

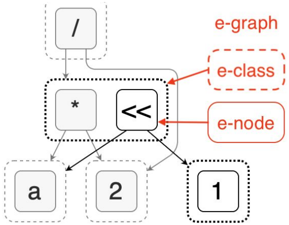
design space exploration and are unable to match skilled engineers. The hardware design space is large and mostly unexplored due to strict correctness requirements and slow debug time frames. The numerous optimization objectives and

: for both humans and automated
synthesis research has focused
tical methods [3], [4], [5], [6] or
7]. Automatic datapath synthesis
oration resulting in better-quality
nprove productivity reducing the
o produce an optimized imple-

piler optimization techniques and
iization [3], [8], we observe that
n at RTL can be described in
:serving transformations. Skilled
mations through experience and
:es of valuable transformations.
be generalized, facilitating their
itomating transformation-driven
nplex since it is often necessary
formations before an ultimately
be applied. Fig. 1 illustrates an
to initially apply transformations
ost via operator duplication or
lead to subsequent area saving
netic simplification or clustering.

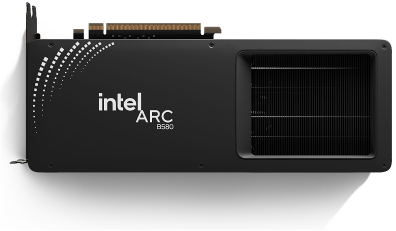
IEEE permission.

E Xplore. Restrictions apply.



ROVER

Numerical HW Group (April 2025)

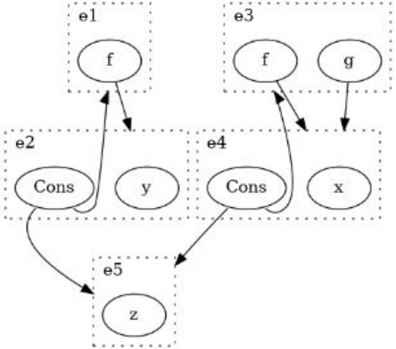


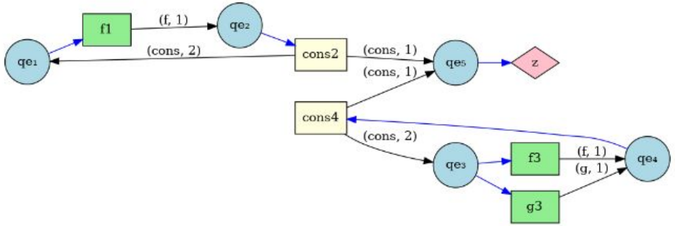
Number Representation:

8-bit: E5M2, E4M3
4-bit: E3M0, E2M1

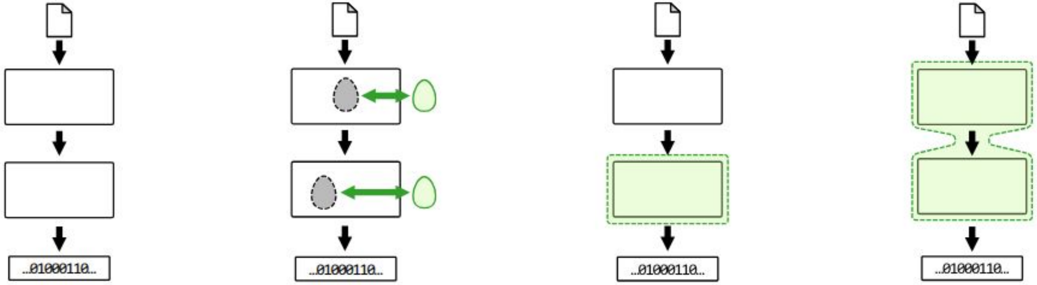
Current Projects

E-Graphs as Automata





Equality Saturation & MLIR



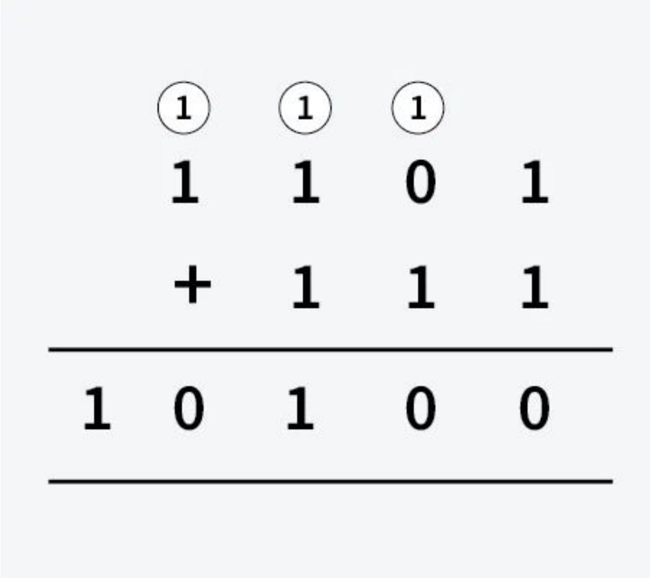
(a) Traditional (b) EqSat Library (c) EqSat Pass (d) EqSat IR

Parametric Bitvector Proofs

$$\forall w_x, w_y\ M_1(x, y) \cong M_2(x, y)$$

Time to talk about circuits…

Addition Is Slow…



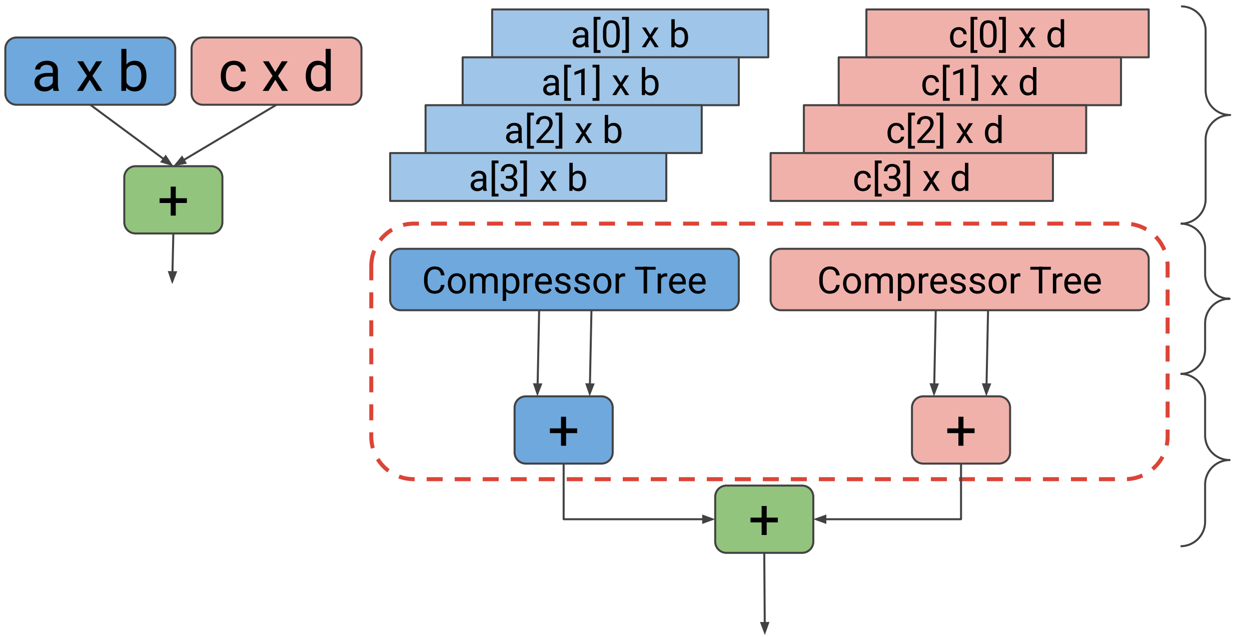
```
   1  1  1
   1  1  0  1
 +    1  1  1
 ─────────────
1  0  1  0  0
 ─────────────
```

```
  1010 = 10 (a)
  1011 = 11 (b)
 +0011 =  3 (c)
  0010 =  2
 1011x = 22
 11000 = 24
```

Reduced 3 values to 2, but we’ve only looked locally - much faster than adding (a+b)

Efficient 4-bit Integer Dot Product Circuit

Naive



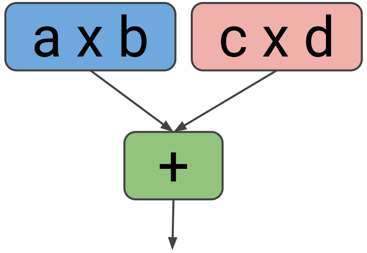
Partial Products

Carry-Save Reduction

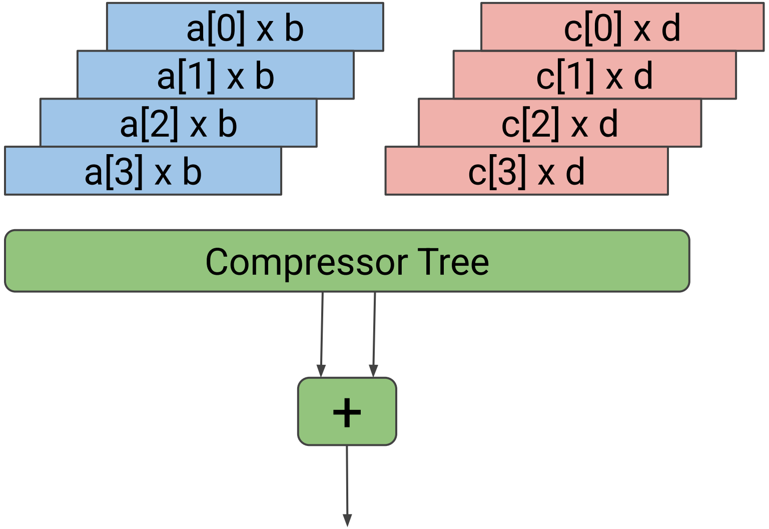
3 = 3 + 0 = 2 + 1 = …

Carry Propagate Adders

Efficient Integer Dot Product Circuit



Efficient



Meh...

4-bit:

➔ 5% faster

➔ 4% smaller

Say more...

16-bit:

➔ 25% faster

➔ 5% smaller

Key Takeaway

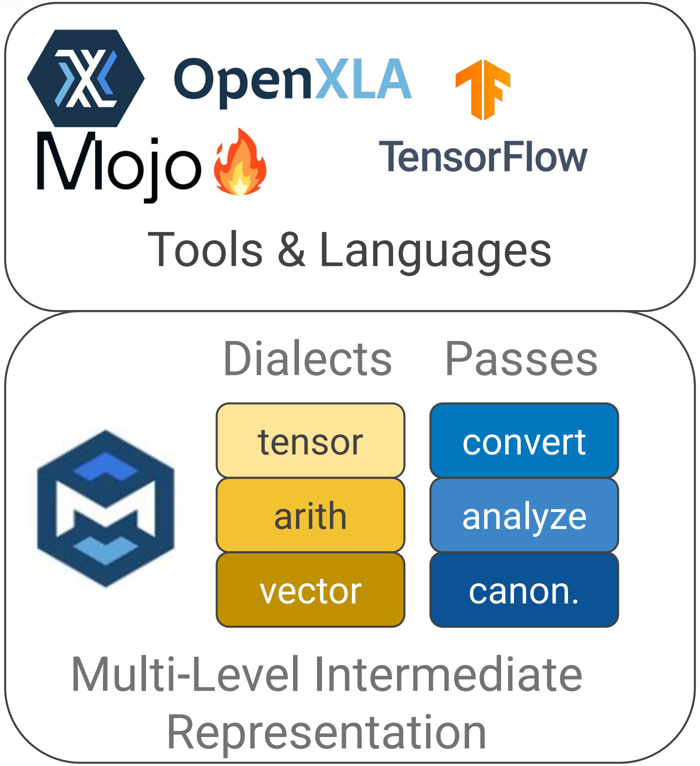
A **hardware intermediate representation** should express **efficient circuit constructs** beyond typical arithmetic operators.

Language Design Target:

- ➔ Composable
- ➔ Verifiable

Datapath Intermediate Representation

MLIR & CIRCT



OpenXLA

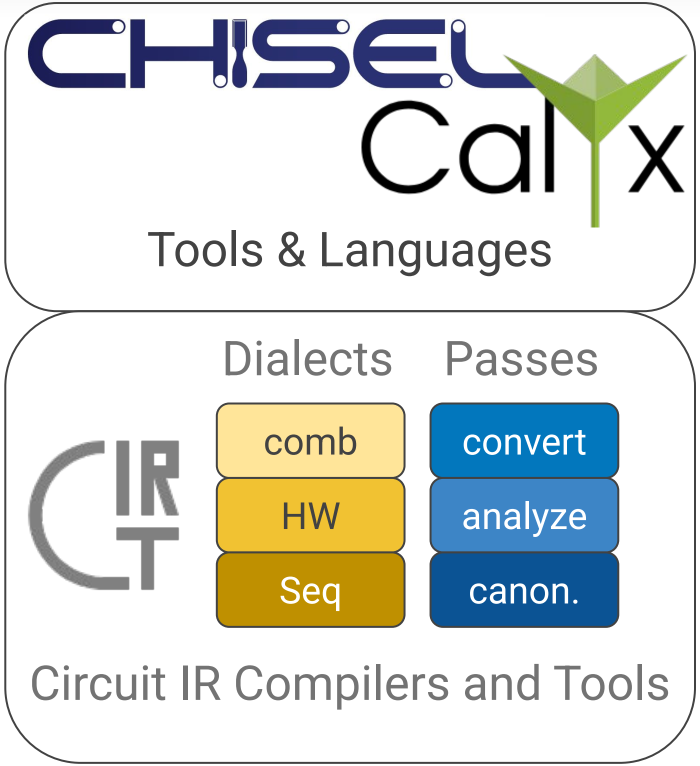
Mojo🔥

TensorFlow

Tools & Languages

Dialects	Passes
tensor	convert
arith	analyze
vector	canon.

Multi-Level Intermediate Representation



CHISEL

Calyx

Tools & Languages

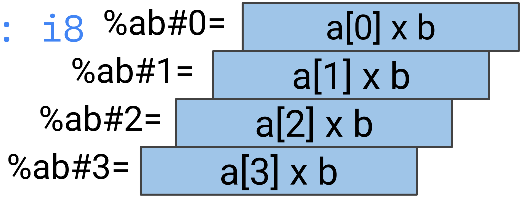
CIRT

Dialects	Passes
comb	convert
HW	analyze
Seq	canon.

Circuit IR Compilers and Tools

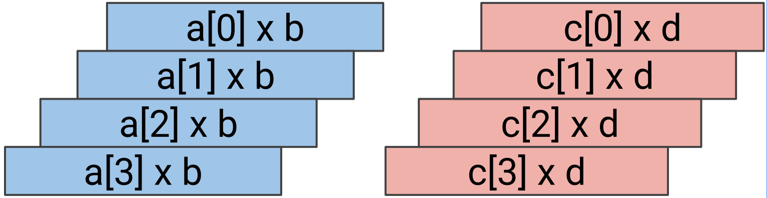
Datapath Dialect: (a x b) + (c x d)

```
%ab:4 = datapath.partial_product %a, %b : i8
```



Datapath Dialect: (a x b) + (c x d)

```
%ab:4 = datapath.partial_product %a, %b : i8
%cd:4 = datapath.partial_product %c, %d : i8
```

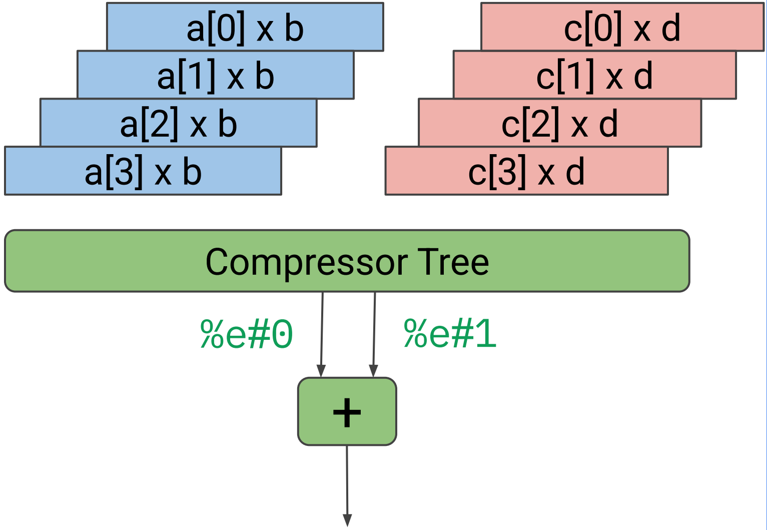


Datapath Dialect: (a x b) + (c x d)

```
%ab:4 = datapath.partial_product %a, %b : i8
%cd:4 = datapath.partial_product %c, %d : i8

// Compress to carry-save form
%e:2 = datapath.compress %ab#0, …, %ab#3,
                         %cd#0, …, %cd#3
                         : i8 [8 -> 2]

// (carry + save) = a*b + c*d
%result = comb.add %e#0, %e#1 : i8
```



Verified via Contracts

Implementation not specified - Booth/AND/…

```
%ab:N = datapath.partial_product %a, %b
// Verification Contract
assert(%ab#0 + … + %ab#<N-1> == %a x %b)
```

Functionality constrained

Verified via Contracts

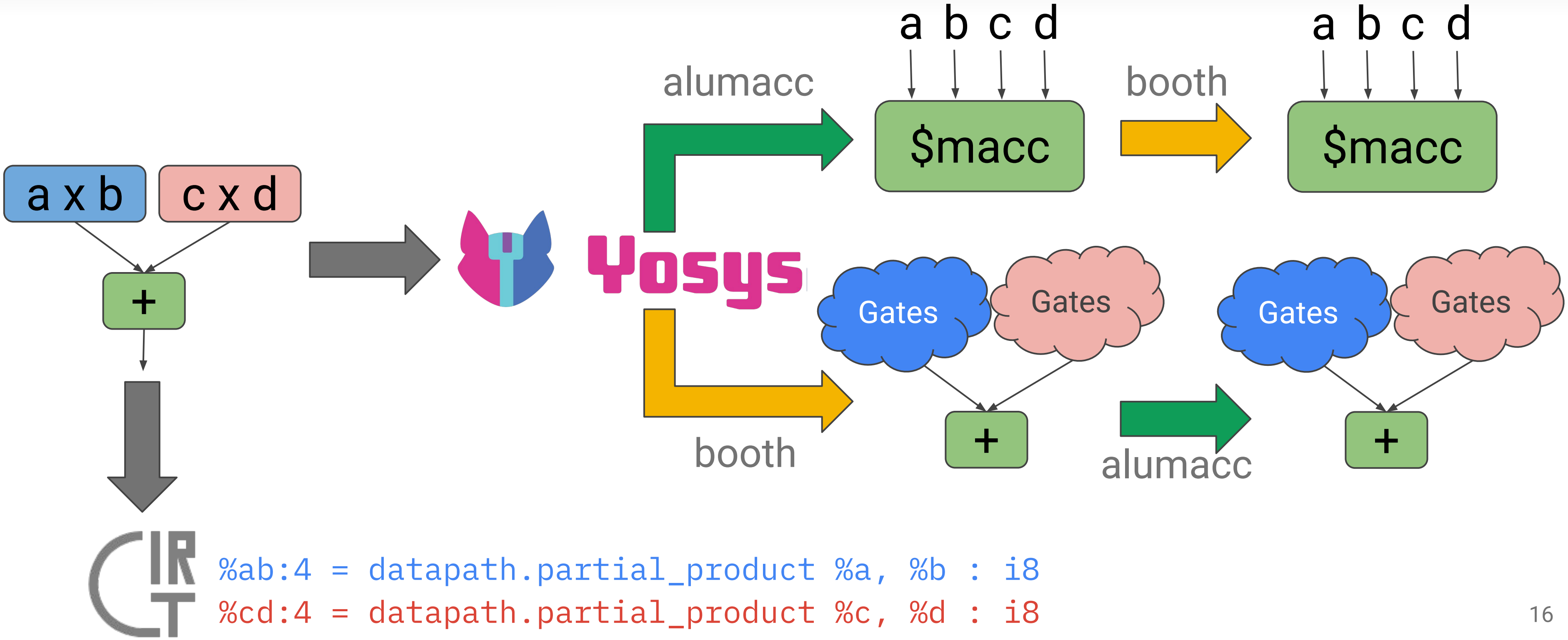
Software:
Undef? Poison?

```
%ab:N = datapath.partial_product %a, %b
// Verification Contract
assert(%ab#0 + … + %ab#<N-1> == %a x %b)
```

```
%z:2 = datapath.compress %y#0, …, %y#<N-1>
// Verification Contract
assert(%z[0] + %z[1] = %y#0 + … + %y#<N-1>)
```

Can you zero-extend y’s?

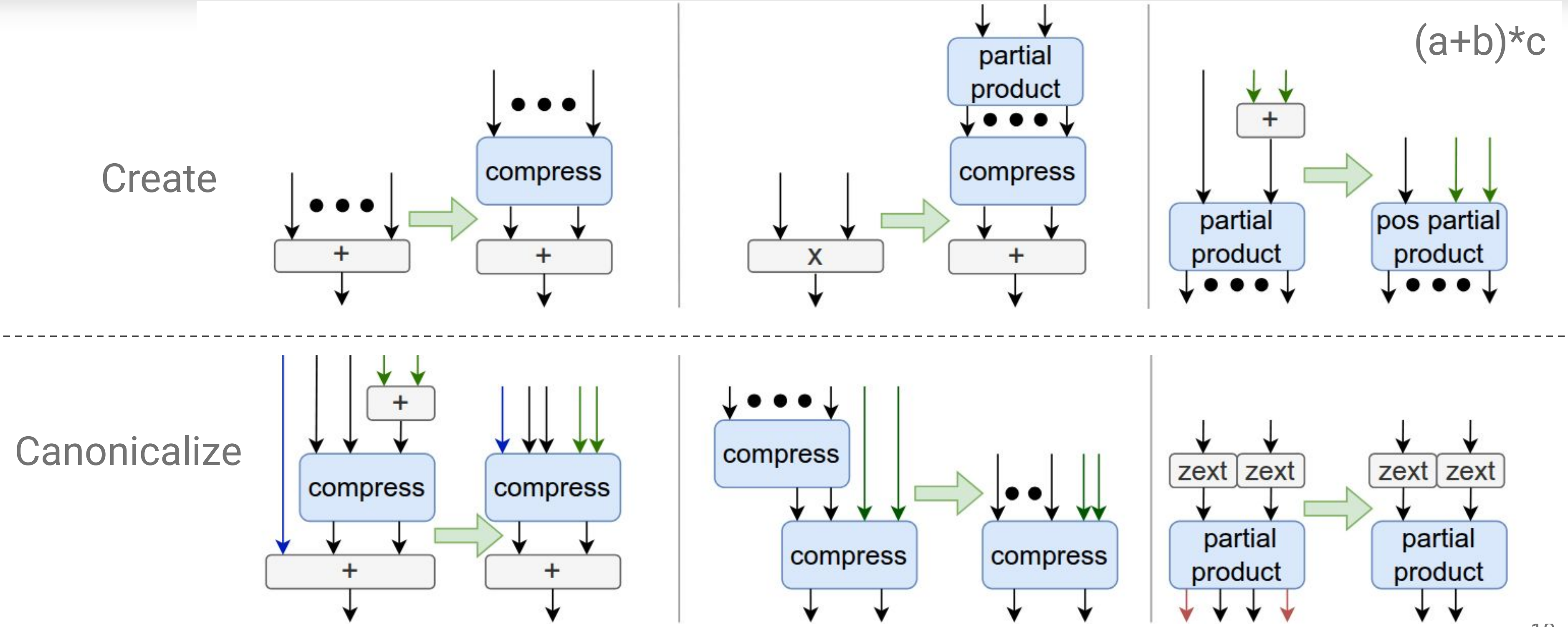
Composability



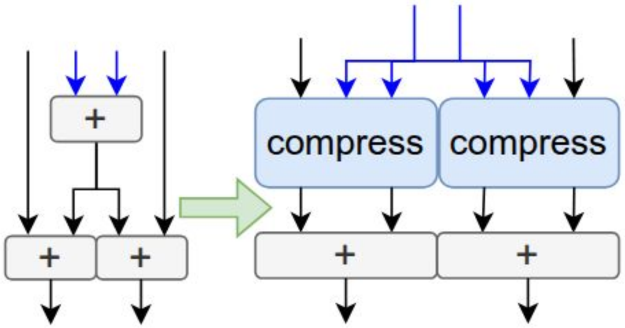
```
%ab:4 = datapath.partial_product %a, %b : i8
%cd:4 = datapath.partial_product %c, %d : i8
```

Automation

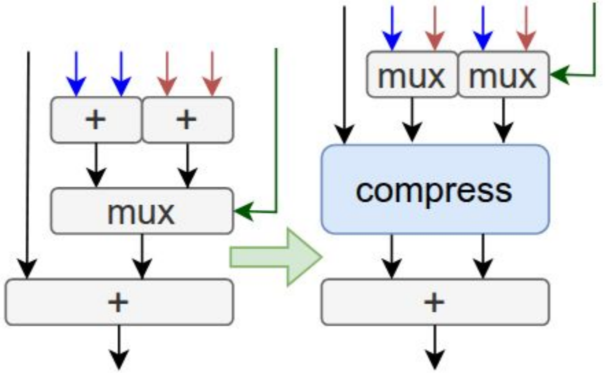
Always Profitable



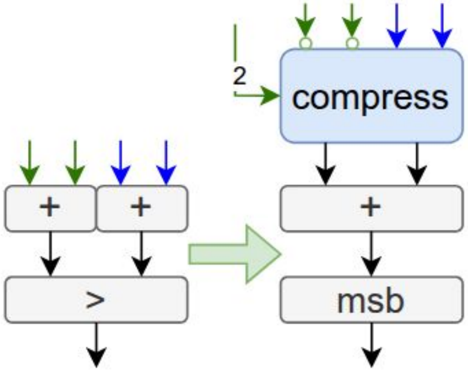
Delay Reducing, Area Increasing



Unsharing

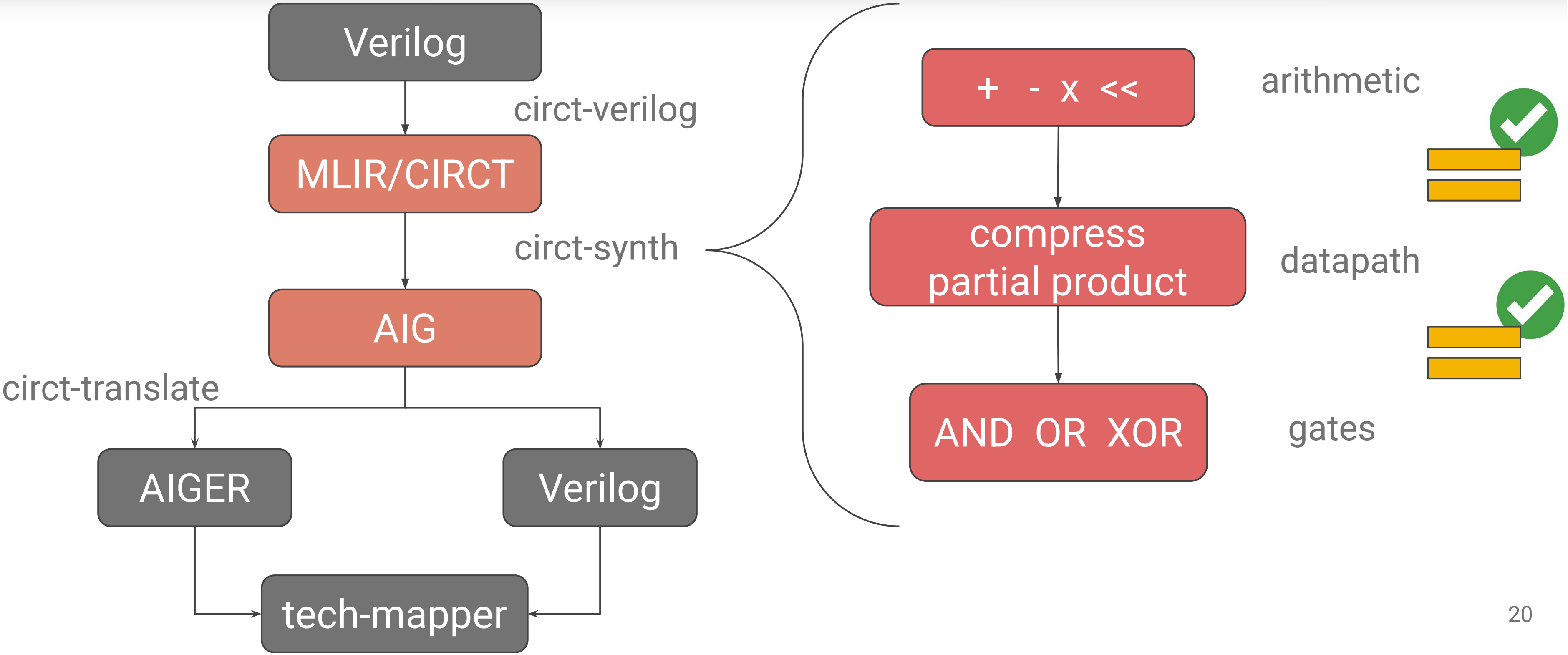


Duplicate Logic

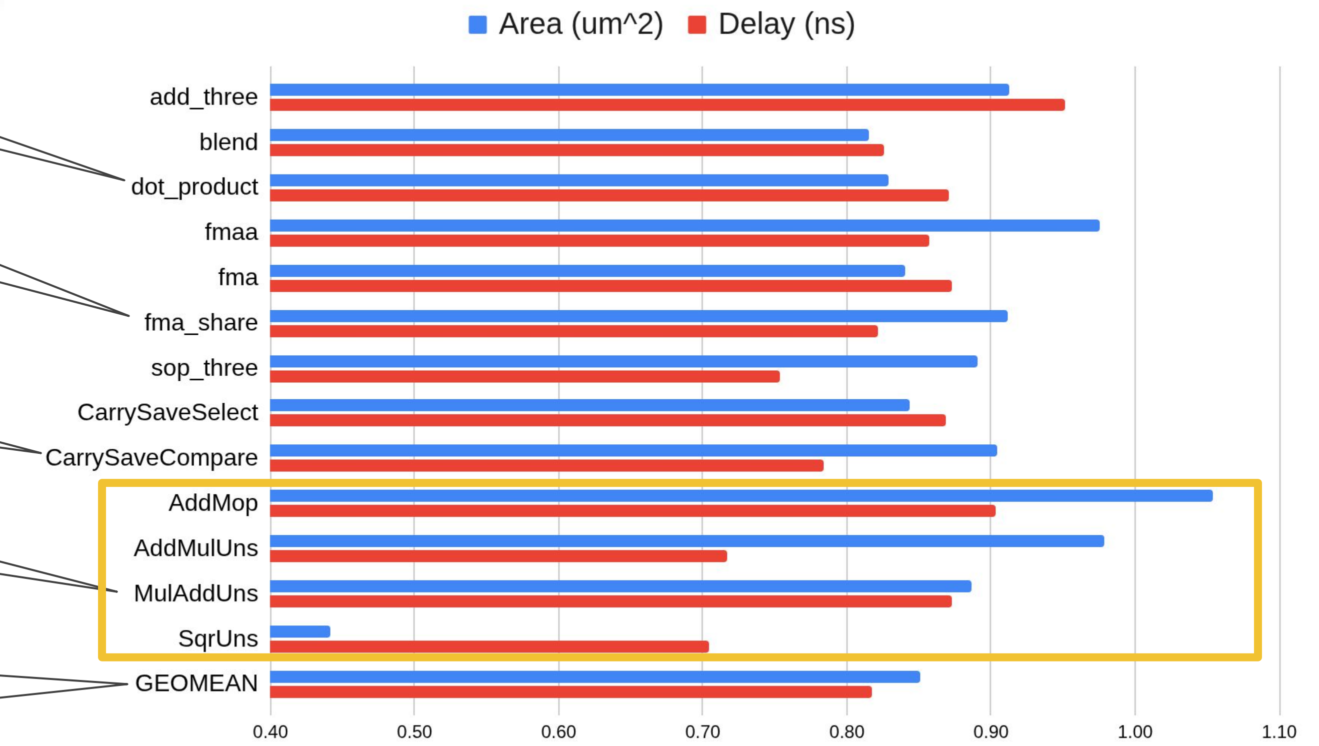


Carry-Save
Comparison

Automated Datapath Synthesis



Comparison vs Yosys (ASAP 7nm)



a*b + c*d

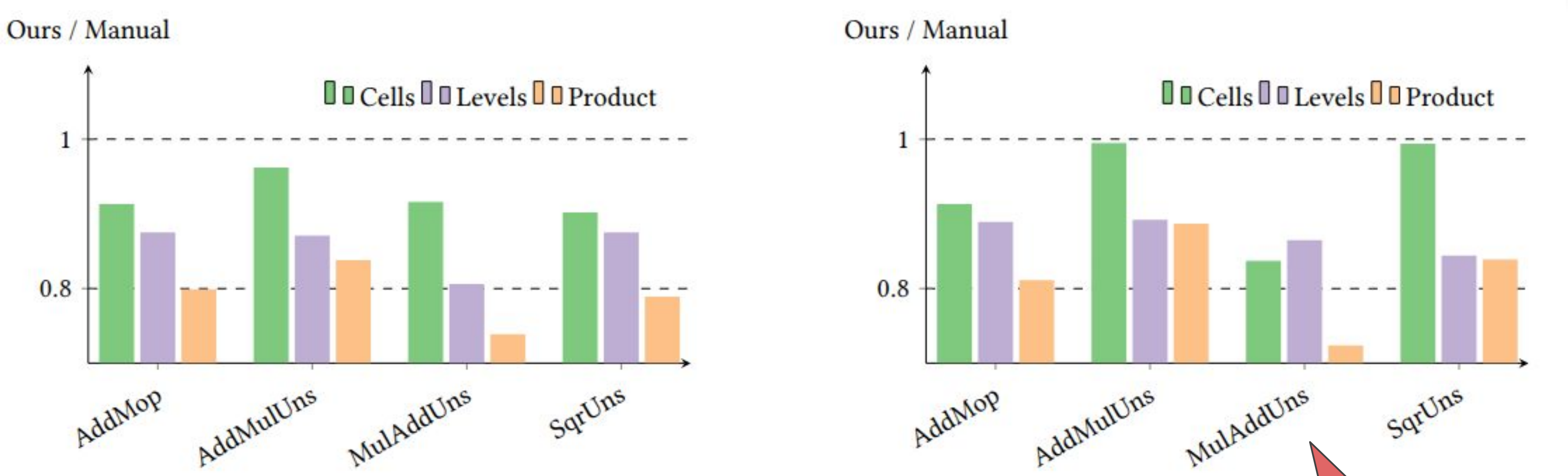
a*b + c
a*b + d

a*b > c+d

a*(b+c)

15% Area
18% Delay

Comparison vs Hand-Optimized (PULP)



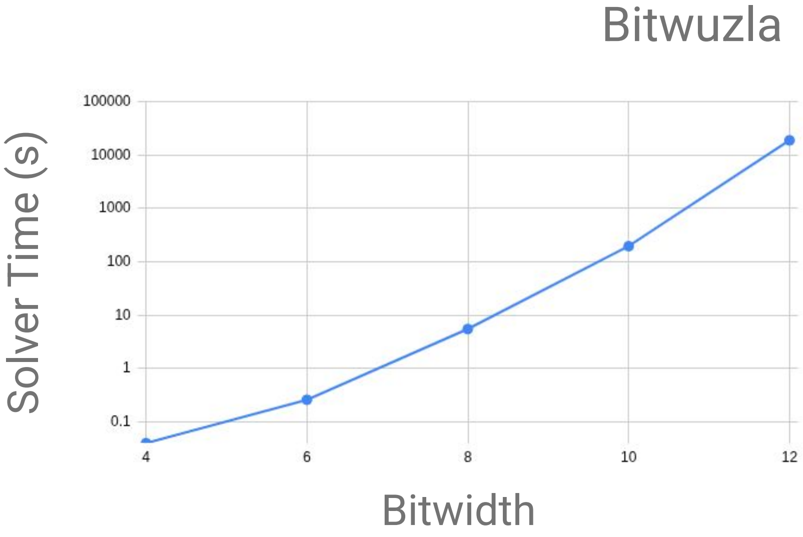
(a) 16-bit inputs

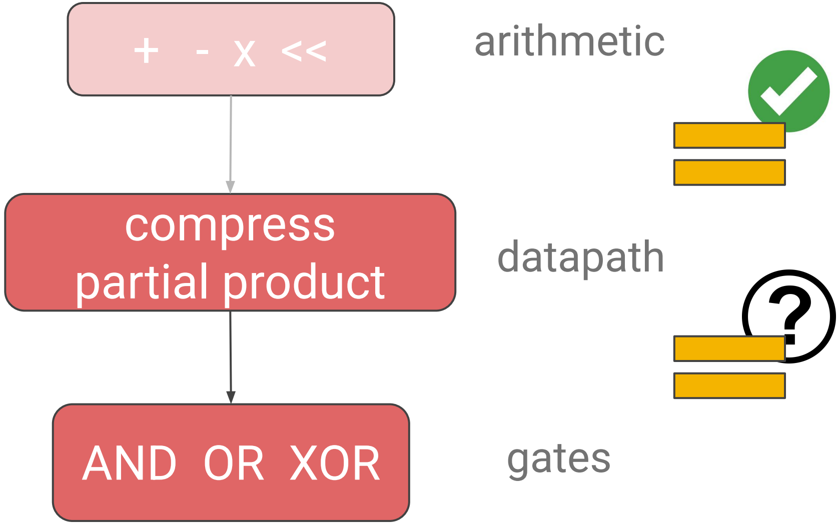
(b) 32-bit inputs

Timing-Aware Compressor Trees

Booth Encoding

Verification Challenges

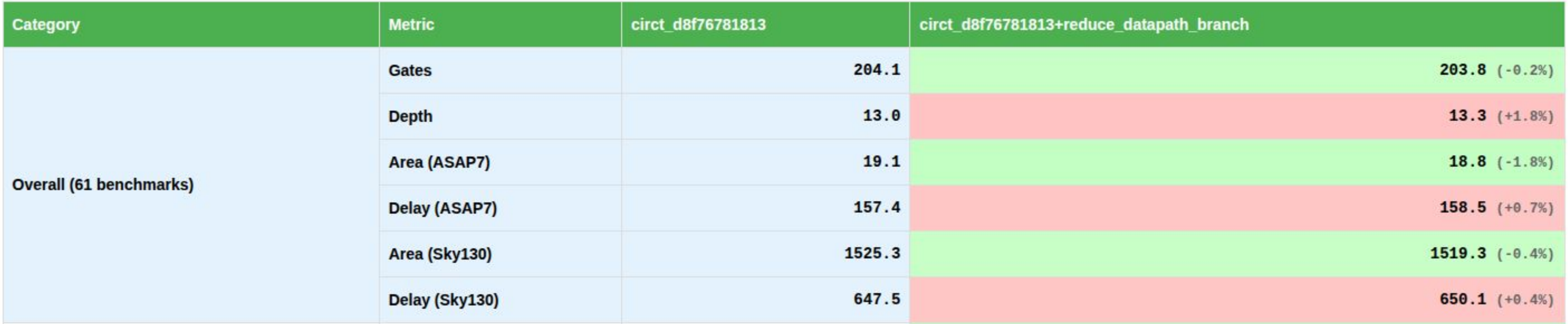




Datapath Benchmarks

- Parameterized System Verilog modules
- Automated logic synthesis competition
- Generates equivalence checking benchmarks

Please contribute your own benchmarks via PRs?



Category	Metric	circt_d8f76781813	circt_d8f76781813+reduce_datapath_branch
Overall (61 benchmarks)	Gates	204.1	203.8 (-0.2%)
	Depth	13.0	13.3 (+1.8%)
	Area (ASAP7)	19.1	18.8 (-1.8%)
	Delay (ASAP7)	157.4	158.5 (+0.7%)
	Area (Sky130)	1525.3	1519.3 (-0.4%)
	Delay (Sky130)	647.5	650.1 (+0.4%)

Conclusion & Next Steps

- Upstream and open-source in CIRCT
- Datapath design benchmarks - share your designs?
- Solver improvements desperately needed…
- Signed arithmetic & pipelining to come

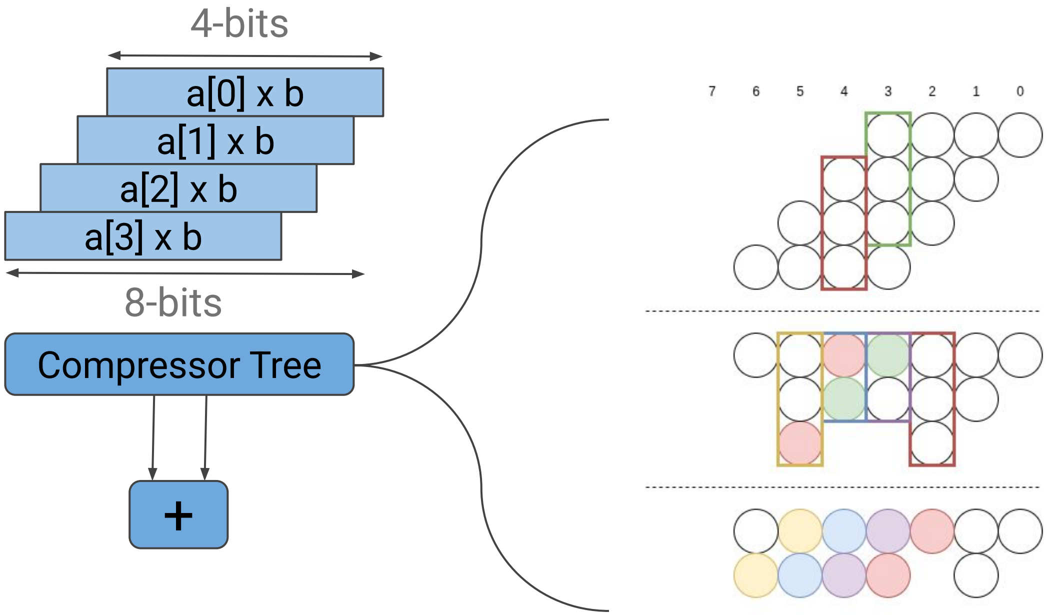
CIRCT

Docs

Benchmarks

BACKUP

BACKUP - Multiplier



4-bits

a[0] x b

a[1] x b

a[2] x b

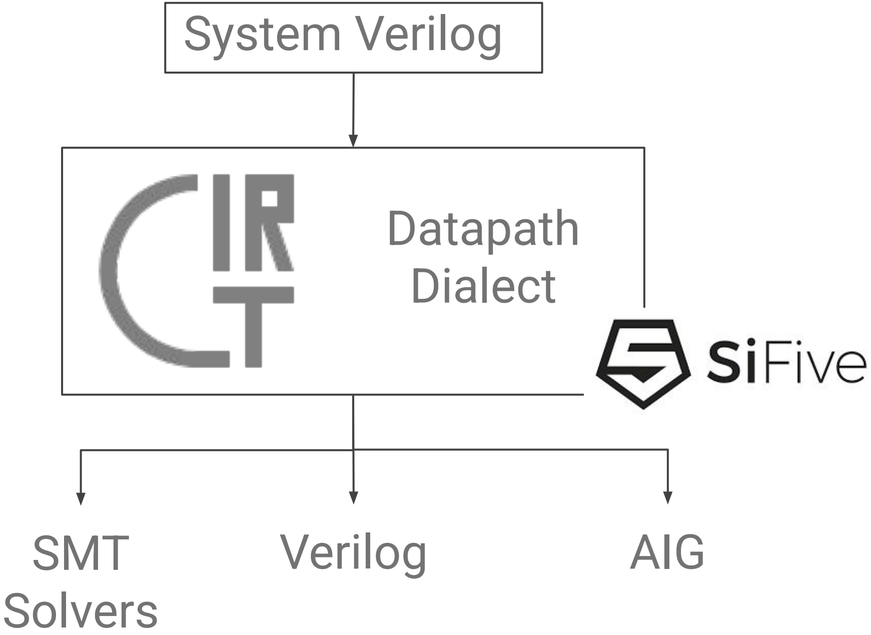
a[3] x b

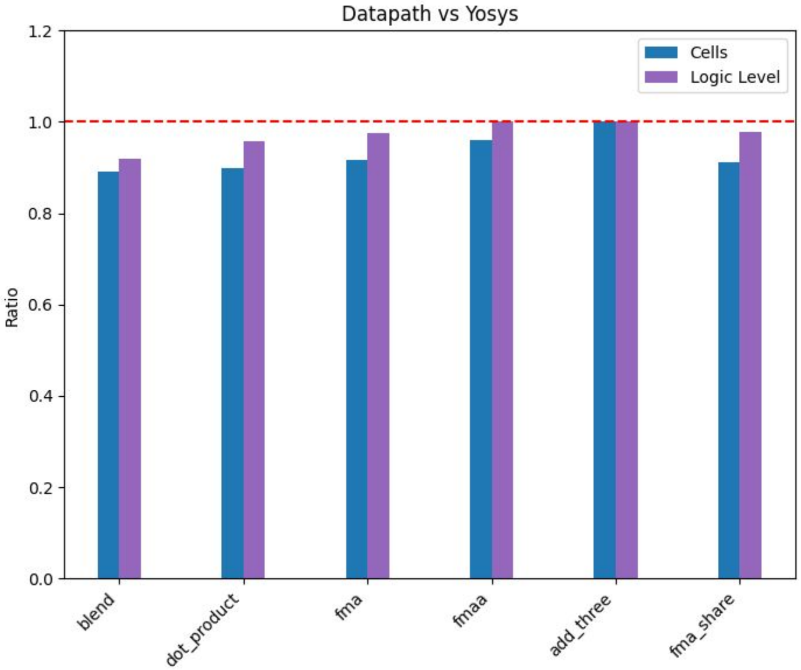
8-bits

Compressor Tree

+

Back-up


System Verilog
CIRCT
Datapath Dialect
SiFive
SMT Solvers
Verilog
AIG


Datapath vs Yosys
Cells
Logic Level
Ratio
1.2
1.0
0.8
0.6
0.4
0.2
0.0
blend
dot_product
fma
fmaa
add_three
fma_share