ASP-DAC 2026
31th Asia and South Pacific Design Automation Conference

CombRewriter: Enabling Combinational Logic Simplification in MLIR-Based Hardware Compiler

Haisheng Zheng[♠] Zhuolun He[♡♣] Shuo Yin[♡] Yuzhe Ma[♠] Bei Yu[♡]

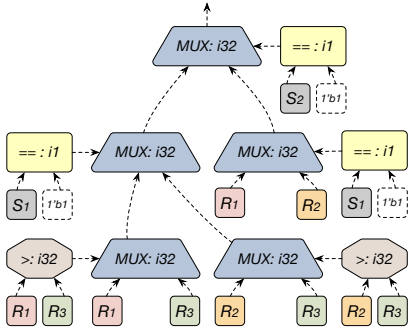
[♠]HKUST (GZ) [♡]CUHK [♣]ChatEDA Tech

Jan. 19, 2026

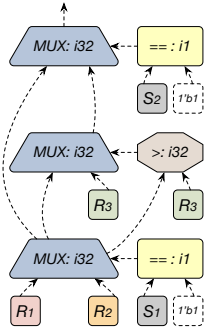
Outline

Introduction

Motivation



(a) Before Simplification.



(b) After Simplification.

A Streamlined Instance Derived From Gemmini [1].

Resource Reduction Using *Simplify Control Flow Graph (SimplifyCFG)* Pass:

- Multiplexers ($5 \rightarrow 3$)
- Greater-Than Comparators ($2 \rightarrow 1$)

[1] **Hasan Genc et al. (2021). "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration".** In: *Proc. DAC*.

Limitations of Applying SimplifyCFG

*(**Example**) Optimizing Resource Usage via Operation Reordering*

- $F = S?(a * b * c) : (a * b * d)$

Transformation	MULT	2-1 MUX
$F = ((S?c : d) * b) * a$	2×16 Bit	16
$F = ((S?c : d) * a) * b$	2×16 Bit	16

(a) Same Bit-Width Case (a=16, b=16, c=16, d=16)

Transformation	MULT	2-1 MUX
$F = ((S?c : d) * b) * a$	2×32 Bit	16
$F = ((S?c : d) * a) * b$	16 Bit ($c \vert d * a$), 32 Bit	16

(b) Different Bit-Width Case (a=16, b=32, c=16, d=16)

- **Transformations** should be specifically tailored for hardware IR to ensure effective logic simplification.

Limitations of Applying SimplifyCFG

*(**Example**) Area Usage Analysis*

- $F = S?(a + b) : c \rightarrow F = (S?a : c) + (S?b : 0) \rightarrow F = (S?a : c) + (S \,\&\, b)$

Resource	Adder	2-1 MUX	AND
Before	64	64	-
After	64	16	64

(a) Better Area Usage
(Bit-Width: a=16, b=64, c=16)

Resource	Adder	2-1 MUX	AND
Before	16	16	-
After	16	16	16

(b) Worser Area Usage
(Bit-Width: a=16, b=16, c=16)

Resource	Adder	2-1 MUX	AND
Before	64	64	-
After	64	48	64

(c) Uncertain Area Usage
(Bit-Width: a=16, b=64, c=48)

- **Performance Analysis** should be specifically designed for hardware IR simplification.

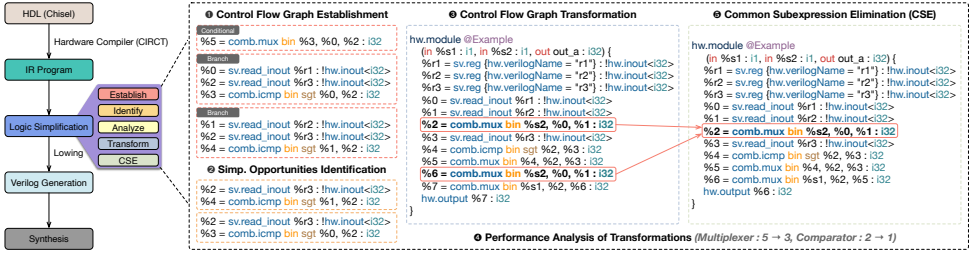
Research Objective

Combinational Logic Simplification

Given a hardware intermediate representation (IR) graph $G = (V, E)$, identify simplification opportunities in the multiplexer branches to optimize G using compiler techniques.

Algorithms

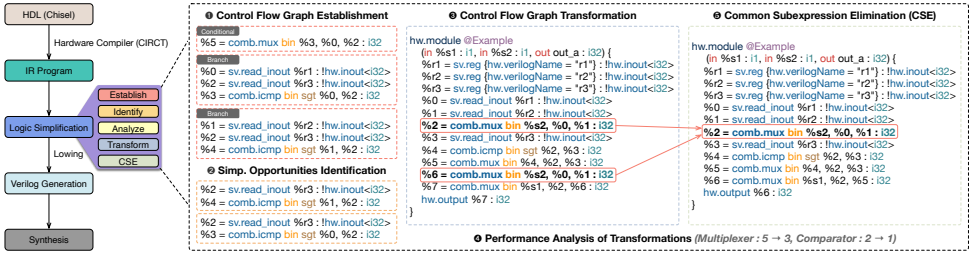
Overview



Transformation Example of Our Proposed Simplification Flow.

1. **Control Flow Graph Establishment**.
   Extract operations in the Hardware IR graph that influence branching and decision-making to build the CFG and establish the order for simplification.

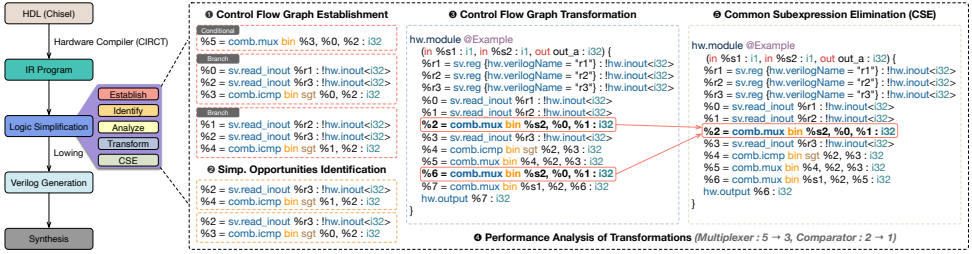
Overview



Transformation Example of Our Proposed Simplification Flow.

1. **Optimization Opportunities Identification**.
   Analyze the control flow to identify opportunities for simplification.

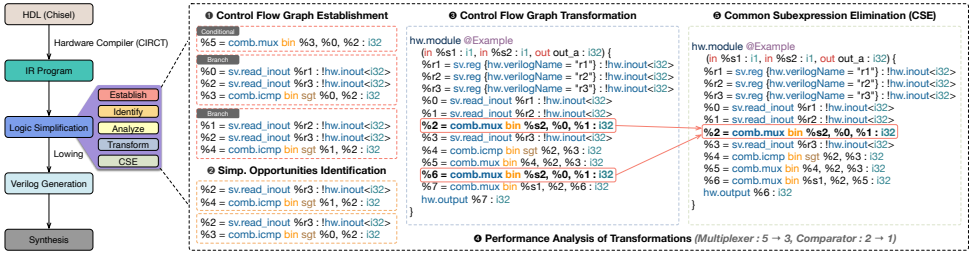
Overview



Transformation Example of Our Proposed Simplification Flow.

1. **Control Flow Graph Simplification**.
   Perform analysis to confirm whether simplifications provide benefits and apply them accordingly.

Overview



Transformation Example of Our Proposed Simplification Flow.

1. **Common Subexpression Elimination (CSE)** is employed to further reduce redundancy, as simplification may introduce common subexpressions.

Control Flow Graph Establishment

*(**Example**) Simplification Order Matters*

- $F = S_1?A \times 3 : (S_2?B \times 3 : C \times 3)$.
- Simplification starting from the *True* branch:
  1. $F = S_1?A \times 3 : (S_2?B : C) \times 3$
  2. $F = S_1?A : (S_2?B : C) \times 3$
- Simplification starting from the *False* branch:
  1. $F = (S_1?A : (S_2?B : C)) \times 3$

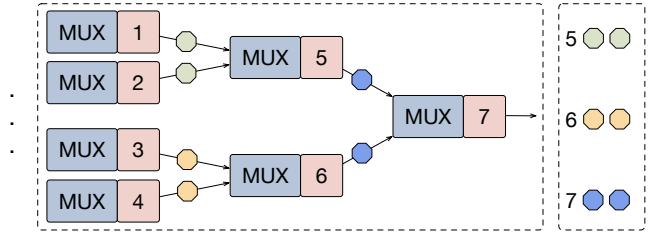
Control Flow Graph Establishment

*(**Example**) Simplification Order Matters*

- $F = S_1?A \times 3 : (S_2?B \times 3 : C \times 3)$.
- Simplification starting from the *True* branch:
  1. $F = S_1?A \times 3 : (S_2?B : C) \times 3$
  2. $F = S_1?A : (S_2?B : C) \times 3$
- Simplification starting from the *False* branch:
  1. $F = (S_1?A : (S_2?B : C)) \times 3$

Determining the *Simplification Order* is crucial for ensuring the efficiency of the simplification process.

Control Flow Graph Establishment



A Visualization of Control Flow Graph.

- *Breadth-First Search (BFS)* is employed to construct the *basic blocks*.
- *Dominator Tree* [2] is employed to determine the simplification order of CFGs.

$$\mathrm{Dom}(n) = \begin{cases} \{n\}, & \text{if } n = n_0 \\ \{n\} \cup \left(\bigcap_{p \in \mathrm{preds}(n)} \mathrm{Dom}(p) \right), & \text{if } n \neq n_0 \end{cases}. \tag{1}$$

[2] **lengauer1979fast.**

Control Flow Graph Simplification & Transformation

* Operations with Fixed Parameters (*CONCAT, EXTRACT, REPLICATE, SHIFT*) Are Non-Gate Operations and Should NOT Introduce XOR/MUX.

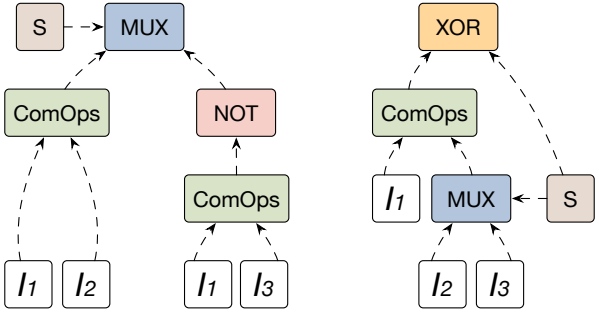
Category	Original	Transformed
MUX Relocation	$F = S\,?\,COMOps(I_1)\ :\ COMOps(I_2)$	$F = COMOps(S\,?\,I_1\ :\ I_2)$
XOR Introduction	$F = S\,?\,COMOps(I_1)\ :\ NOT(COMOps(I_2))$	$F = \mathrm{XOR}(COMOps(S\,?\,I_1\ :\ I_2),\ S)$
MUX Introduction	$F = S\,?\,DIFFOps_1(COMOps(I_1))\ :\ DIFFOps_2(COMOps(I_2))$	$F = S\,?\,DIFFOps_1(F_{COM})\ :\ DIFFOps_2(F_{COM}),$ $F_{COM} = COMOps(S\,?\,I_1\ :\ I_2)$

Common Operation Hoisting Transformations.

Category	Original	Transformed
Arithmetic Logic Exchange	$F = S?(a+b) : c$	$F = (S?a : c) + (S?b : 0)$
	$F = S?a : (b+c)$	$F = (S?a : b) + (S?1 : c)$
	$F = (S?0 : a) \times b$	$F = a \times (S?0 : b)$
Operation Reordering	$F = ((S?c : d) * b) * a$	$F = ((S?c : d) * a) * b$
Shift Operation Distribution	$F = (S?a : c) \ll (S?b : d)$	$F = S?(a \ll b) : (c \ll d)$
	$F = (S?a : c) \gg (S?b : d)$	$F = S?(a \gg b) : (c \gg d)$

Domain-Specific Transformations (The $*$ Operation Represents Both $+$, $\times$).

Simplification Opportunities Identification



(a) Original. (b) Transformed.

Introducing XOR Operation to Simplify CFG.

- Formulated as a *Graph Isomorphism* problem.
- A *VF2-Based Algorithm* [3] is designed to identify simplification opportunities.

[3] **Luigi Pietro Cordella et al. (2001). "An Improved Algorithm for Matching Large Graphs". In:** *IAPR-TC15 International Workshop on Graph-Based Representations in Pattern Recognition*.

ML-Assisted Performance Analysis: Motivation

High-Level Descriptions Hinder Precise Estimation of Resource Usage

- Uncertainty in area usage estimation.
- Impact of simplification on timing performance.

[4] **Haoyuan Wu et al. (2025). "Circuit Representation Learning with Masked Gate Modeling and Verilog-AIG Alignment".** In: *Proc. ICLR*.

[5] **Ceyu Xu et al. (2023). "Fast, Robust and Transferable Prediction for Hardware Logic Synthesis".** In: *Proc. MICRO*.

[6] **Yanqing Zhang, Haoxing Ren, and Brucek Khailany (2020). "GRANNITE: Graph Neural Network Inference for Transferable Power Estimation".** In: *Proc. DAC*.

ML-Assisted Performance Analysis: Motivation

High-Level Descriptions Hinder Precise Estimation of Resource Usage

- Uncertainty in area usage estimation.
- Impact of simplification on timing performance.

Several studies[4][5][6] *employ graph neural networks (GNNs) to*

- Extract circuit features → Predict performance.

[4] **Haoyuan Wu et al. (2025). "Circuit Representation Learning with Masked Gate Modeling and Verilog-AIG Alignment".** In: *Proc. ICLR*.

[5] **Ceyu Xu et al. (2023). "Fast, Robust and Transferable Prediction for Hardware Logic Synthesis".** In: *Proc. MICRO*.

[6] **Yanqing Zhang, Haoxing Ren, and Brucek Khailany (2020). "GRANNITE: Graph Neural Network Inference for Transferable Power Estimation".** In: *Proc. DAC*.

SimplifyGuider: GNN-Assisted Performance Analysis

- **Node Feature Encoding**
  - Operation Information (Operation Type & its Operand Width)
  - Node Level Encoding:

$$\mathrm{PE}(v) = \sin\left(\frac{level(v)}{L_{\max}}\right) + \cos\left(\frac{level(v)}{L_{\max}}\right) \tag{2}$$

SimplifyGuider: GNN-Assisted Performance Analysis

- **Node Feature Encoding**
  - Operation Information (Operation Type & its Operand Width)
  - Node Level Encoding:

$$\mathrm{PE}(v) = \sin\left(\frac{level(v)}{L_{\max}}\right) + \cos\left(\frac{level(v)}{L_{\max}}\right) \tag{2}$$

- **Hardware IR Graph Feature Extraction**
  - GraphSAGE:

$$\boldsymbol{h}_v^{(k+1)} = \mathrm{MEAN}^{(k)}\left(\{\boldsymbol{h}_u^{(k)} \mid u \in \mathcal{N}(v)\}\right) \oplus \boldsymbol{h}_v^{(k)} \tag{3}$$

SimplifyGuider: GNN-Assisted Performance Analysis

- **Node Feature Encoding**
  - Operation Information (Operation Type & its Operand Width)
  - Node Level Encoding:

$$\mathrm{PE}(v) = \sin\left(\frac{level(v)}{L_{\max}}\right) + \cos\left(\frac{level(v)}{L_{\max}}\right) \tag{2}$$

- **Hardware IR Graph Feature Extraction**
  - GraphSAGE:

$$\boldsymbol{h}_v^{(k+1)} = \mathrm{MEAN}^{(k)}\left(\{\boldsymbol{h}_u^{(k)} \mid u \in \mathcal{N}(v)\}\right) \oplus \boldsymbol{h}_v^{(k)} \tag{3}$$

- **Pairwise Ranking Loss**

$$L_{\text{rank}} = -\log\left(\sigma(\boldsymbol{f}(g_1) - \boldsymbol{f}(g_2))\right) \tag{4}$$

Experiments

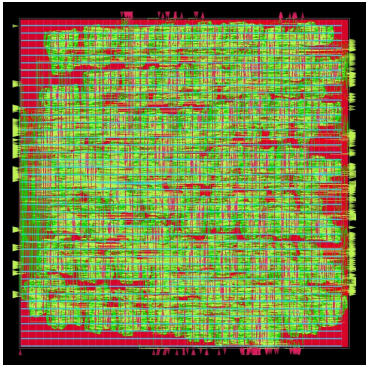
Evaluation Results

Table: Comparison of E-morphic [7] and *CombRewriter* in Netlist Optimization.

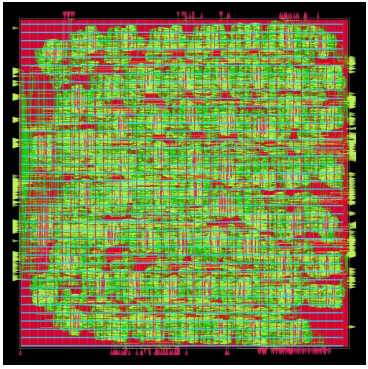
Category	E-morphic[?]		***CombRewriter* (Ours)**	
	$\Delta Area$ (%)	$\Delta Delay$ (%)	$\Delta Area$ (%)	$\Delta Delay$ (%)
MUX Reduction	-10.50%	-8.08%	-47.64%	-13.97%
XOR Introduction	-3.75%	-2.15%	-45.06%	-12.94%
MUX Introduction	-4.73%	-9.08%	-44.62%	-10.81%
Arithmetic Logic Exchange	+2.86%	+2.79%	-4.25%	-1.11%
Operation Reordering	-8.02%	-8.58%	-23.12%	-16.83%
Shift Operation Distribution	-8.71%	+26.00%	-4.64%	-8.54%

[7] **Chen Chen et al. (2025). "E-morphic: Scalable Equality Saturation for Structural Exploration in Logic Synthesis".** In: *Proc. DAC*.

Impact of Simplification on Gemmini [1]



(a) P&R Result w/o *CombRewriter*.



(b) P&R Result with *CombRewriter*.

Method	Standard Cells	Area (μm^2)	Clock Skew (ps)	Worst Slack (ns)	Route Length
Original	200,178	23,861.50	51.94	0.795	730,358
CombRewriter	161,649	19,485.10	49.26	1.089	577,600
Improvement	19.25%	18.34%	5.16%	36.98%	20.92%

[1] **Hasan Genc et al. (2021). "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration". In: *Proc. DAC*.**

Conclusion

- An approach for addressing logic simplification from a **Compiler Design Perspective**.
- **Domain-Specific Transformations** designed to exploit hardware IR-specific simplification opportunities.
- A **Machine Learning-Assisted Performance Analysis Approach**, specifically tailored for hardware logic simplification.

THANK YOU!