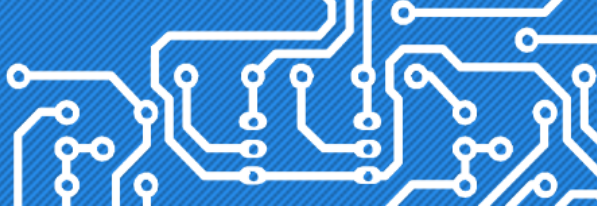
International Symposium on Physical Design



HeLO: A Heterogeneous Logic Optimization Framework by Hierarchical Clustering and Graph Learning

Yuan Pu, Fangzhou Liu, Zhuolun He, Keren Zhu, Rongliang Fu,
Ziyi Wang, Tsung-Yi Ho, Bei Yu

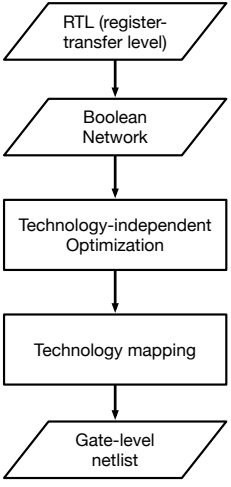
Department of Computer Science and Engineering
The Chinese University of Hong Kong

Outline

Introduction

Logic Synthesis Flow

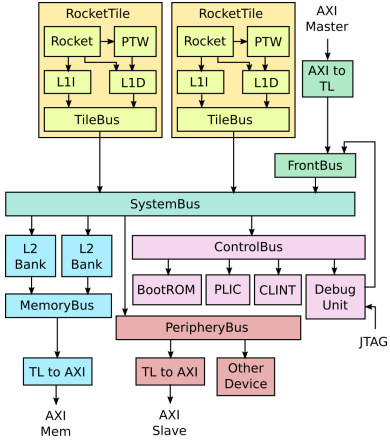
- RTL design: a high-level design abstraction modelling digital circuits.
- Boolean network: directed acyclic graph (DAG).
  - directed acyclic graph, node for Boolean function (AND, OR, ect.) and edge for wire.
  - DAG types:
    - and-inverter graph (**AIG**): AND gates and inverter on the edge.
    - majority-inverter graph (**MIG**): 3-input majority node.
    - **XAG**, **XMG**, etc.
  - Technology-independent logic optimization:
    - minimize node count (area) and depth (critical path).
    - multi-level optimization strategy: rewrite, rebalance.
    - Trade-off for node count and depth.
    - Metric: node-depth product (NDP).



Heterogeneous Logic Optimization: Motivation

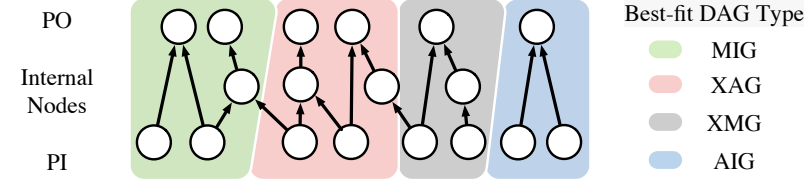
- Modern VLSI designs such as SoC, CPU ⇒ consists of modules with various structures and functionalities
- Different modules ⇒ different optimization strategies/scripts for logic optimization ⇒ Optimal PPA.

Rocket SoC Diagram.



Heterogeneous Logic Optimization: Introduction

- Represent different circuit portions by different DAG types; Optimize separately (by corresponding optimization strategy).
- Motivations:
  - Different DAG types → different expressive power for specific structures/functionalities.
  - Different DAG types → different optimization algorithms desired.
  - Conclusion: Different DAG types → different logic optimization results.
- Example: **MIG** effectively represents carry operators, thus leading to better optimization result for arithmetic designs.



Previous Work for Hetero Logic Optimization

- MixSyn[1]:
  - Detect AND/OR and XOR components in a circuit.
  - Render optimization separately.
  - Drawbacks: Hard to extend to other gate types.
- LSOracle[2]:
  - min-cut circuit partition (k-way partition) into sub-circuits.
  - CNN to predict best-fit DAG type for subcircuit (treating the Karnaugh-map of subcircuit as image).
  - Render logic optimization for each sub-circuit, based on predicted DAG type.

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli (2013). "MIXSyn: An efficient logic synthesis methodology for mixed XOR-AND/OR dominated circuits". In: *Proc. ASPDAC*, pp. 133–138.

[2] Walter Lau Neto, Max Austin, et al. (2019). "LSOracle: A logic synthesis framework driven by artificial intelligence". In: *Proc. ICCAD*, pp. 1–6.

Limitations of LSOracle

- partitions consist of logic components with varying structures and functions → hard to determine best-fit DAG type.
- Treat best-fit DAG type prediction as an image classification task → lacks utilization of topological information.

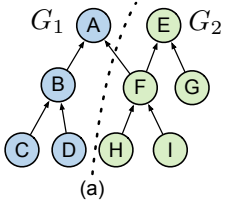
Motivation

Observation 1

- **O1**: When two sub-circuits exhibit **similarities** in their structures or functionalities, they often select the same DAG type for optimal results.
- Explanation:
  - Efficacy of different DAG types varies with specific Boolean functions and topological structures.
  - Circuits with similar functions or topologies opt for the same DAG type to maximize expressive power and optimization outcomes.

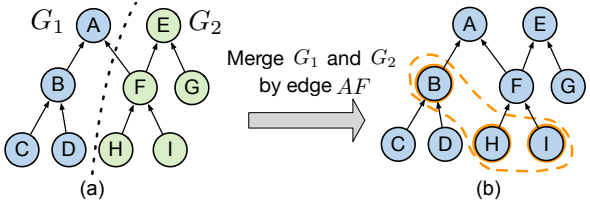
Observation 2

- **O2**: If two interconnected sub-circuits are functionally or structurally similar, merging them into a single circuit and performing logic optimization often yields better results than optimizing each separately.
- Explanation:
  - Multi-level optimization algorithms : cut-based.
  - Merging two circuits $\rightarrow$ new cut choices generated at intersection $\rightarrow$ enlarging the solution space.



Observation 2

- **O2**: If two interconnected sub-circuits are functionally or structurally similar, merging them into a single circuit and performing logic optimization often yields better results than optimizing each separately.
- Explanation:
  - Multi-level optimization algorithms : cut-based.
  - Merging two circuits $\rightarrow$ new cut choices generated at intersection $\rightarrow$ enlarging the solution space.

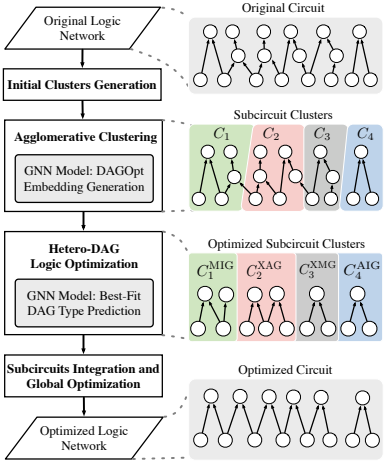


Proposed Algorithm

Overall Flow

Main techniques:

- **Agglomerative clustering**: allocate structurally/functionally similar logic components into the same sub-circuit.
- **GNN model**:
  - Generate structural-functional embeddings for sub-circuits.
  - Determine best-fit DAG type.
- Hetero-DAG logic optimization:
  - Support four DAG types: AIG, MIG, XMG, XAG.
  - Optimization: script provided by LSOracle.



Overcome Limitations of LSOracle

- partitions consist of logic components with varying structures and functions → hard to determine best-fit DAG type.
- Solution: Agglomerative clustering.
- Treat best-fit DAG type prediction as an image classification task → lacks utilization of topological information.
- Solution: GNN for determining best-fit DAG type.

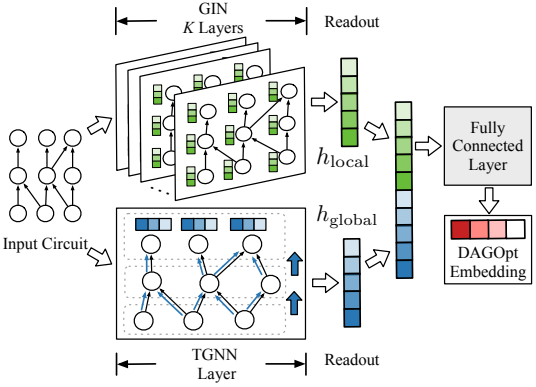
DAGOpt Embedding Space

- Purpose:
  - Quantify the structural and functional similarities across various circuits.
  - Used for agglomerative clustering.
- Construction embedding e^c for a circuit c:
  - Represent c into AIG, MIG, XMG, XAG.
  - Render logic optimization on the four representations and obtain corresponding node-depth product (NDP).
  - Compilie the NDP values into a normalized vector.
- Two structurally/functionally similar sub-circuits $\rightarrow$ similar DAGOpt embeddings.

$$e^c = \frac{(\mathrm{NDP}^{\mathrm{AIG}}, \mathrm{NDP}^{\mathrm{MIG}}, \mathrm{NDP}^{\mathrm{XMG}}, \mathrm{NDP}^{\mathrm{XAG}})}{\sum_{t \in \{\mathrm{AIG},\ \mathrm{MIG},\ \mathrm{XMG},\ \mathrm{XAG}\}} \mathrm{NDP}^t}. \tag{1}$$

Customized GNN Model

- graph isomorphism model (GIN):
  - Map structurally similar sub-graphs to similar embeddings.
  - Capture local structural information.
  - Read out (mean) node embeddings into h_{local}.
- topological graph neural network(TGNN):
  - Message passing following topological order.
  - Readout (mean) node embeddings of POs (exclude POs of small connected components) into h_{global}.
  - Mimic logic simulation, capture global structural and functional information.



Agglomerative Clustering

Steps:

- step 1: Initial cluster generation: treat each fanin cone of PO as an initial cluster.
- step 2: Pre-trained GNN $\rightarrow$ generate DAGOpt embedding for each initial cluster, as its coordinate.
- step 3: **Merge** two connected clusters with the most similar embedding into one sub-circuit.
- step 4: Repeat step 3 until some termination condition.

Reasons for choosing PO fanin cone as initial cluster:

- Fanin cone of a PO encompasses every potential cut for its nodes $\rightarrow$ preserving complete cut-based structural information.
- Logic function at PO only depends on the logic outputs of all leave nodes in the PO fanin cone $\rightarrow$ preserving independent functional information.

Illustration of Agglomerative Clustering

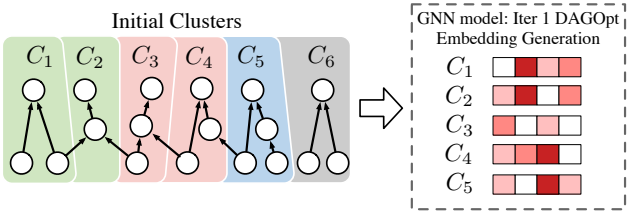


Illustration of Agglomerative Clustering

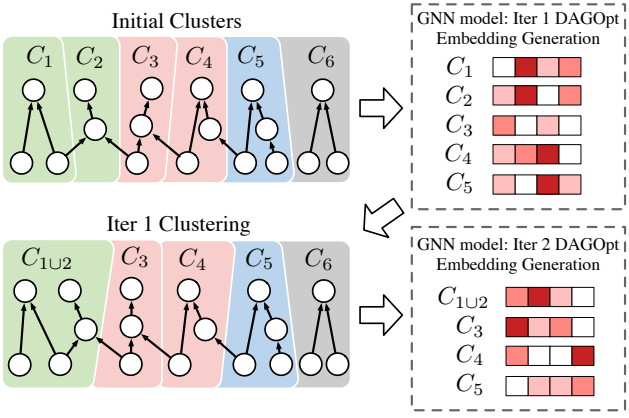
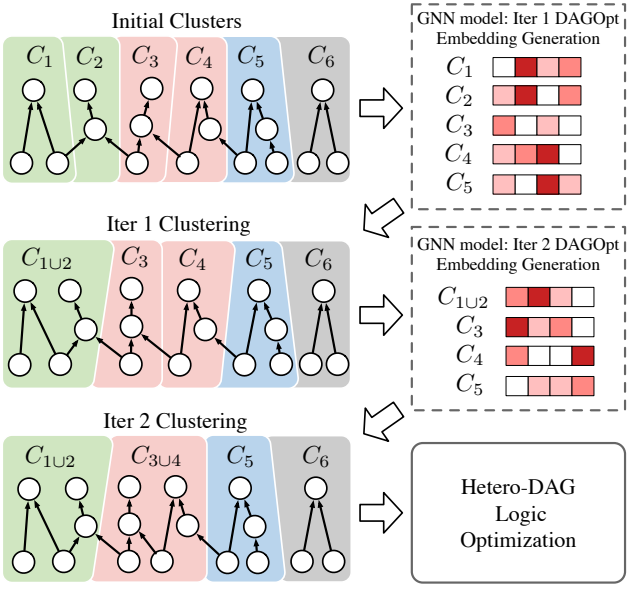


Illustration of Agglomerative Clustering



Hetero-DAG Logic Optimization

- Best-fit DAG type prediction:
  - Pre-trained GNN $\rightarrow$ generate DAGOpt embedding e^c.
  - DAG type corresponding to smallest value in e^c $\rightarrow$ best-fit DAG type.
- Optimization: optimize each sub-circuit by predicted DAG type, using script provided by LSOracle.
- Integration: Convert sub-circuits into MIGs (with no change of node count and depth).
  - MIGs $\supset$ AIGs: setting one input of 3-majority gate to 1 $\rightarrow$ AND gate.
  - MIGs $\supset$ XMG.
  - MIGs $\supset$ XAG: setting one input of 3-majority gate to 1, invert two input edges $\rightarrow$ XOR gate.
- Global optimize: rewrite.

Experimental Results

Dataset Preparation

- 5714 sub-circuits selected from EPFL, ISCAS'89 benchmark suites and OpenCores.
- Script of LSOracle to obtain the DAGOpt embedding of each sub-circuit, as ground truth.
- Distribution of sub-circuits favoring AIG, MIG, XMG, and XAG as the best-fit DAG type $\rightarrow 0.52 : 0.28 : 0.16 : 0.30$.
- Note: some circuits might have more than one best-fit DAG types.

Evaluation

Evaluation of prediction accuracy:

- Training/testing split ratio: 80:20.
- Accuracy for best-fit DAG type prediction: 79.99%.

Evaluation of hetero-DAG optimization:

- Baseline: ABC[3], FlowTune[4], Mockturtle[5], LSOracle.
- Both **technology-independent logic optimization** and **technology mapping**.
- pdk: ASAP7.

[3]Robert Brayton and Alan Mishchenko (2010). "ABC: An academic industrial-strength verification tool". In: *Proc. CAV*, pp. 24–40.

[4]Walter Lau Neto, Yingjie Li, et al. (2022). "FlowTune: End-to-end Automatic Logic Optimization Exploration via Domain-specific Multi-armed Bandit". In: *IEEE TCAD*.

[5]Mathias Soeken et al. (2018). "The EPFL logic synthesis libraries". In: *arXiv preprint arXiv:1805.05121*.

Technology-Independent Logic Optimization Results

Table: Technology-independent logic toptimization result. NDP denotes the product of node count and depth.

Circuit	Original			ABC (30*resyn)			Flowtune			Mockturtle			LSOracle			HeLO (ours)		
	#nodes	depth	NDP	#nodes	depth	NDP	#nodes	depth	NDP	#nodes	depth	NDP	#nodes	depth	NDP	#nodes	depth	NDP
pico-rv	18139	31	562309	15775	30	473250	14641	52	761332	20036	21	420756	18838	21	395598	19268	18	**346824**
chip_bridge	58789	31	1822459	57733	26	1501058	56377	23	1296671	59237	19	1125503	59538	19	1131222	58317	19	**1108023**
s38417	8568	28	239904	7842	24	188208	7730	22	170060	9016	18	162288	9028	18	162504	9522	16	**152352**
fpu	66522	33	2195226	58477	31	1812787	56731	33	1872123	66889	23	1538447	67248	22	1401752	68099	20	**1361980**
aes_core	21522	26	559572	19822	20	396440	19302	22	424644	20825	21	437325	21561	27	582147	21867	18	**393606**
des_perf	72720	16	1163520	72394	16	1158304	56593	16	**905488**	70176	15	1052640	70176	15	1052640	70176	15	1052640
ethernet	69763	41	2860283	66443	34	2259062	65684	31	2036204	70226	25	1755650	68482	24	1643568	71896	20	**1437920**
dyn_node	3926	27	106002	3620	24	86880	3596	22	79112	3979	19	**75601**	4191	21	88011	4034	18	72612
DMA	4295	20	85900	3450	17	58650	3301	19	**62719**	4348	15	65220	4208	17	71536	4342	15	65130
vga_lcd	105828	22	2328216	103583	21	2175243	103191	21	2167011	107657	16	**1722512**	108465	16	1735440	101534	17	1726078
fpga_bridge	318195	42	13364190	315998	37	11691926	301337	36	10848132	340217	26	8845642	325698	27	8793846	324356	24	**7784544**
i2c	1342	20	26840	1047	14	14658	1009	11	11099	1417	9	12753	1387	11	15257	1385	8	**11080**
mem_ctrl	46836	114	5339304	43608	104	4535232	36366	81	**2945646**	51762	69	3571578	52123	68	3544364	56592	61	3452112
normalize	1.000	1.000	1.000	0.967	0.882	0.860	0.911	0.863	0.769	1.037	0.656	0.678	1.018	0.678	0.673	1.019	0.596	**0.619**

- Node-depth product (NDP) of HeLOis reduced by 38.9%, 24.3%, 9.6% and 8.7%, compared with the result of ABC(30*resyn), FlowTune, Mockturtle and LSOracle.

Technology Mapping Result

Table: ASIC technology mapping result using the ASAP7 PDK. Area is in um^2 and delay is in ps. ADP denotes the product of delay and area.

Circuit	Original			ABC (30*resyn)			Flowtune			Mockturtle			LSOracle			HeLO (ours)		
	area	delay	ADP	area	delay	ADP	area	delay	ADP	area	delay	ADP	area	delay	ADP	area	delay	ADP
pico-rv	775.5	439.1	340492.0	779.2	434.5	338563.3	764.3	680.6	520193.6	841.5	312.9	263316.0	778.2	312.9	243503.0	831.6	290.0	**241153.0**
chip_bridge	3016.0	310.1	935394.0	3097.2	325.8	1008896.6	3010.2	308.3	928117.2	2988.2	294.2	879253.0	3038.9	294.2	894162.0	3028.2	263.9	**799010.0**
s38417	418.6	280.3	117319.0	415.4	304.4	126459.9	415.8	302.0	125572.7	415.7	283.0	117648.0	432.3	267.8	115776.0	416.2	266.7	**111016.0**
fpu	3115.4	466.5	1453351.0	3134.1	467.5	1465111.8	3101.4	523.3	1622869.6	3111.5	455.9	1418551.0	3062.2	452.2	1384682.0	3127.9	324.8	**1016029.0**
aes_core	1032.6	280.0	289164.0	1019.9	293.6	299428.1	941.5	320.3	301562.5	1033.1	277.5	286670.0	1001.5	291.2	291683.0	1061.0	251.0	**266276.0**
des_perf	3325.8	242.2	805408.0	3647.0	265.4	968020.6	3114.1	267.8	833899.1	3457.8	232.8	**804807.0**	3457.8	232.8	**804807.0**	3457.8	232.8	**804807.0**
ethernet	3476.7	384.6	1337324.0	3424.2	470.4	1610860.5	3495.8	465.4	1627101.5	3486.5	309.6	1079382.0	3366.8	306.5	1031939.0	3407.0	289.4	**985948.0**
dyn_node	204.4	293.2	59925.0	198.7	317.8	63156.4	203.9	286.7	58450.0	201.7	266.3	53703.0	212.0	251.5	53321.0	205.5	231.7	**47610.0**
DMA	182.6	196.9	35963.6	178.7	198.1	**35394.3**	179.1	212.4	38041.2	185.3	206.9	38350.3	186.1	196.7	36611.9	185.8	196.7	36556.3
vga_lcd	6125.5	300.0	1837402.0	5374.8	262.4	1410242.6	5491.7	306.7	1684097.0	5896.3	237.8	1401913.8	5751.7	237.8	**1367531.3**	5627.5	259.5	1460459.2
fpga_bridge	17049.5	584.6	9967166.9	16578.4	499.6	8281754.7	15978.7	580.2	9271167.1	17053.6	331.0	5644403.8	16760.3	356.9	5982606.9	16385.7	340.9	**5585048.8**
i2c	57.7	247.3	14278.5	50.4	230.0	11597.4	50.9	161.1	8201.7	59.5	131.7	**7840.7**	59.7	133.0	7940.7	60.2	131.7	7931.6
mem_ctrl	2282.5	1559.8	3560243.5	2165.5	1496.9	3241553.5	1850.2	1191.1	**2203815.5**	2333.1	1086.0	2533863.3	2340.3	1086.0	2541704.5	2395.2	1021.3	2446214.2
normalize	1.000	1.000	1.000	0.976	0.997	0.909	0.940	1.004	0.926	1.000	0.792	0.700	0.985	0.791	0.711	0.979	0.734	**0.665**

- Area-delay product (ADP) of HeLOis reduced by 36.6%, 39.2%, 5.2% and 6.9%, compared with the result of ABC, FlowTune, Mockturtle and LSOracle.

Runtime Comparison

Table: Runtime analysis of ABC, Flowtune, Mockturtle, LSOracle and HeLO for logic optimization. The unit of the runtime is second (s).

Circuit	ABC (30*resyn)	Flowtune	Mockturtle	LSOracle	HeLO (ours)
pico-rv	15	324	29	113	57
chip_bridge	62	541	792	1616	262
s38417	7	81	3	56	30
fpu	86	665	28	633	480
aes_core	23	166	41	120	63
des_perf	121	530	93	680	70
ethernet	70	1007	1276	5106	803
dyn_node	3	20	23	18	12
DMA	3	24	2	28	59
vga_lcd	128	9174	3002	12460	3254
fpga_bridge	3032	9160	72461	70789	47611
i2c	1	12	7	11	21
mem_ctrl	56	876	61	414	337
Normalize.	0.068	0.426	1.467	1.735	1.000

- ABC: very fast; only provide AIG optimization; least effective optimization result.
- FlowTune: multi-armed bandit exploration, slower than ABC.
- Mockturtle: slow for optimizing whole large Boolean Network.
- LSOracle: slow for enumerating each DAG-type optimization for each sub-circuit.

THANK YOU!