ASP-DAC 2024
29th Asia and South Pacific Design Automation Conference

LSTP : A Logic Synthesis Timing Predictor

Haisheng Zheng[1], Zhuolun He[1,2], Fangzhou Liu[1,2],
Zehua Pei[1,2], Bei Yu[2]

[1]Shanghai AI Laboratory, Shanghai, China
[2]The Chinese University of Hong Kong

Jan. 25, 2024

Outline

Introduction

Background

Logic Synthesis is critical:

- Architecture exploration relies on the acquisition of metrics reported by logic synthesis[1]
- Logic synthesis quality determines the best possible design space of subsequent procedure[2]

[1] Chen Bai et al. (2021). "BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework". In: *Proc. ICCAD*.

[2] Ceyu Xu et al. (2022). "SNS's Not a Synthesizer: A Deep-Learning-Based Synthesis Predictor". In: *Proc. ISCA*.

Logic Synthesis is critical:

- Architecture exploration relies on the acquisition of metrics reported by logic synthesis[1]
- Logic synthesis quality determines the best possible design space of subsequent procedure[2]

Can we efficiently predict the desired metrics without actually running expensive logic synthesis?

[1] Chen Bai et al. (2021). "BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework". In: *Proc. ICCAD*.

[2] Ceyu Xu et al. (2022). "SNS's Not a Synthesizer: A Deep-Learning-Based Synthesis Predictor". In: *Proc. ISCA*.

Previous Works

Work	Estimation	Algorithm
D-SAGE[3]	Timing	GNN
Yu *et al.*[4]	Timing, Area	LSTM
PowerNet[5]	Dynamic IR Drop	CNN
GRANNITE[6]	Power	GNN
Deep H-GCN[7]	Analog Circuit Degradation (i.e., aging)	GNN
De *et al.*[8]	Timing	ML methods
SNS	Timing, Area, Power	Transformer

[3]Ecenur Ustun et al. (2020). "Accurate operation delay prediction for FPGA HLS using graph neural networks". In: *Proc. ICCAD*.

[4]Cunxi Yu et al. (2020). "Decision making in synthesis cross technologies using LSTMs and transfer learning". In: *Proc. MLCAD*.

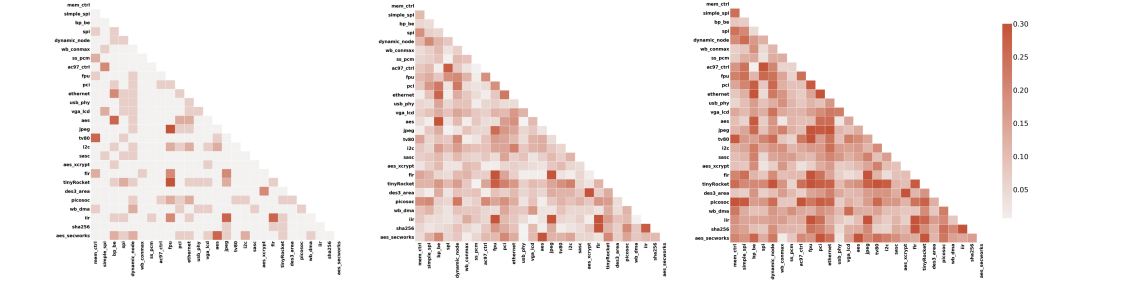
[5]Zhiyao Xie et al. (2020). "PowerNet: Transferable dynamic IR drop estimation via maximum convolutional neural network". In: *Proc. ASPDAC*.

[6]Yanqing Zhang et al. (2020). "GRANNITE: Graph neural network inference for transferable power estimation". In: *Proc. DAC*.

[7]Tinghuan Chen et al. (2021). "Deep H-GCN: Fast analog IC aging-induced degradation estimation". In: *IEEE TCAD*.

[8]Sayandip De et al. (2022). "Delay Prediction for ASIC HLS: Comparing Graph-based and Non-Graph-based Learning Models". In: *IEEE TCAD*.

Logic Synthesis Recipes are NOT One-size-fits-all



(a) Top 1% (b) Top 5% (c) Top 10%

- OpenABC-D[9] has pointed out quantitatively that the similarity between the best synthesis recipes for a set of benchmark circuits is less than 30%.

[9] Animesh Basak Chowdhury et al. (2021). "OpenABC-D: A large-scale dataset for machine learning guided integrated circuit synthesis". In: *arXiv preprint arXiv:2110.11292*.

Previous Works

- Yu *et al.*[10] propose to train a CNN to predict the quality of an optimization sequence.
- Reinforcement Learning (RL) is leveraged[11,12] to generate fixed-length optimization sequences.

[10] Cunxi Yu et al. (2018). "Developing synthesis flows without human knowledge". In: *Proc. DAC*.

[11] Winston Haaswijk et al. (2018). "Deep learning for logic optimization algorithms". In: *Proc. ISCAS*.

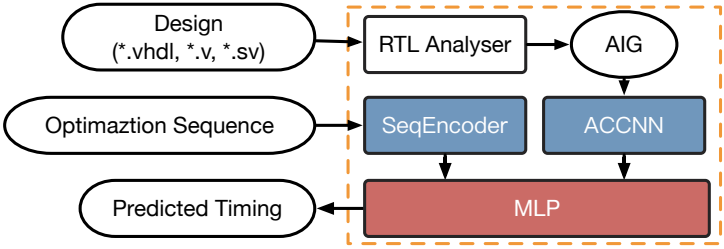
[12] Keren Zhu et al. (2020). "Exploring logic optimizations with reinforcement learning and graph convolutional network". In: *Proc. MLCAD*.

Logic Synthesis Timing Prediction

Given a gate-level netlist as an And-Inverter Graph (AIG) representing a set of Boolean functions and a sequence of subgraph optimization procedures for the AIG graph, design a novel learning methodology that automatically predicts the `final timing` after applying the optimization procedures to the AIG.

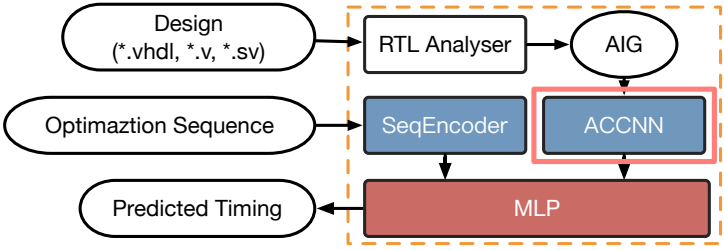
Algorithm

LSTP Overall Flow



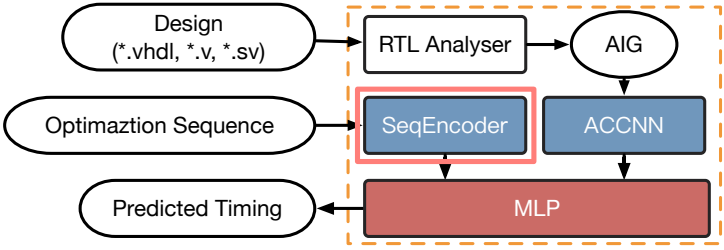
- **RTL-Analyzer** compiles the input design and transforms it into an And-Inverter-Graph (AIG) representation.

LSTP Overall Flow



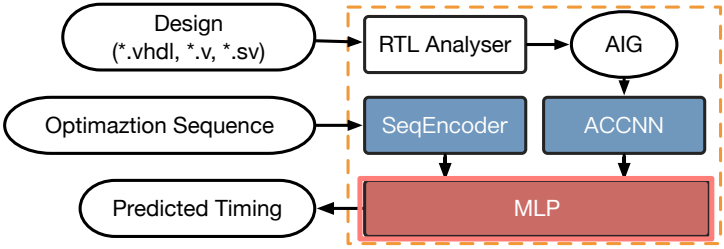
- **RTL-Analyzer** compiles the input design and transforms it into an And-Inverter-Graph (AIG) representation.
- **ACCNN** is a trained graph neural network (GNN) for node sampling and feature extraction of the AIG circuit.

LSTP Overall Flow



- **RTL-Analyzer** compiles the input design and transforms it into an And-Inverter-Graph (AIG) representation.
- **ACCNN** is a trained graph neural network (GNN) for node sampling and feature extraction of the AIG circuit.
- **SeqEncoder** is a trained Transformer encoder for optimization sequence features extraction.

LSTP Overall Flow



- **RTL-Analyzer** compiles the input design and transforms it into an And-Inverter-Graph (AIG) representation.
- **ACCNN** is a trained graph neural network (GNN) for node sampling and feature extraction of the AIG circuit.
- **SeqEncoder** is a trained Transformer encoder for optimization sequence features extraction.
- **MLP** aggregates both the optimization sequence features and the circuit diagram features to predict the timing of the input design.

RTL Analyser

HDL Code

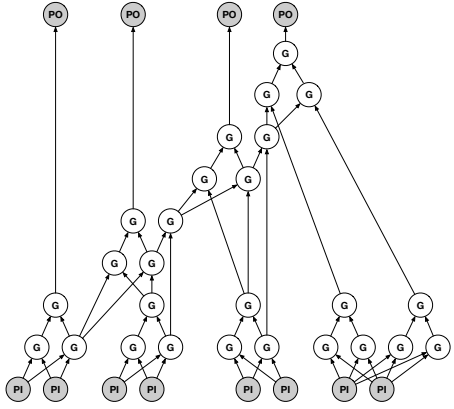
```
module Subtractor
(
  input   [3:0] in0,
  input   [3:0] in1,
  output  [3:0] out
);

assign out = in0 - in1;

endmodule
```

Yosys

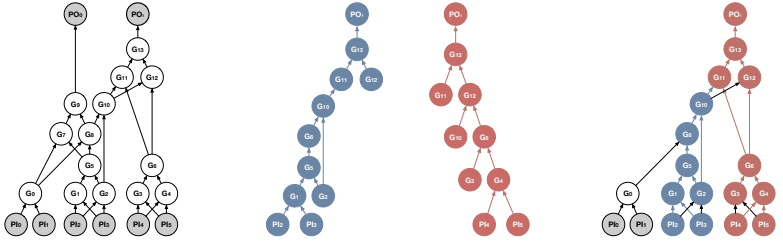
And-Inverter-Graph (AIG)



Asynchronous Cascaded-Cone Neural Network (ACCNN)

- The delay of a circuit depends on the number of hops on the longest path from the primary inputs(PIs) to the primary outputs (POs).
- We wish to design an algorithm to effectively exploit the characteristics of graph representation for the longest path in AIG.

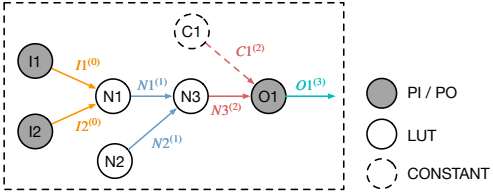
A visual illustration of *sampling cascaded cones*.



A random walk-based approach to sample *cascaded cones* within the circuit

- Each 'path' originate from primary input (PI) and end at primary output (PO)
- The output of flip-flop $\rightarrow$ PI
- The input of flip-flop $\rightarrow$ PO

ACCNN: Netlist Feature Extraction



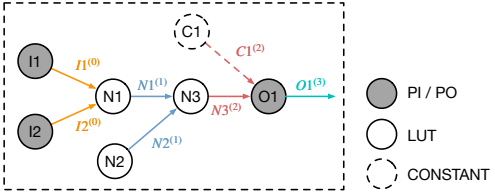
	I1	I2	N1	N2	N3	C1	O1
T = 0	✓	✓					
T = 1			✓	✓			
T = 2					✓	✓	
T = 3							✓

A visual illustration of *ACCNN*.

- We aim for a model that similar to logic simulation, efficiently propagating information step-by-step along the sampled paths
- ABGNN[13] serves this purpose well

[13] Zhuolun He et al. (2021). "Graph Learning-Based Arithmetic Block Identification". In: *Proc. ICCAD*.

ACCNN: Netlist Feature Extraction



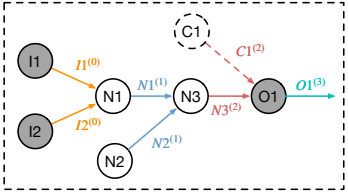
	I1	I2	N1	N2	N3	C1	O1
T = 0	✓	✓					
T = 1			✓	✓			
T = 2					✓	✓	
T = 3							✓

A visual illustration of *ACCNN*.

$$a^{(k)}_{\{i:\mathcal{D}(i,v)=\Delta-k\}} = \text{AGGREGATE}(\{h^{(k-1)}u : u \in N(i)\}) \tag{1}$$

$$h^{(k)}_{\{i:\mathcal{D}(i,v)=\Delta-k\}} = \text{COMBINE}(a_i^{(k)}, h_i^{(0)}) \tag{2}$$

ACCNN: Netlist Feature Extraction



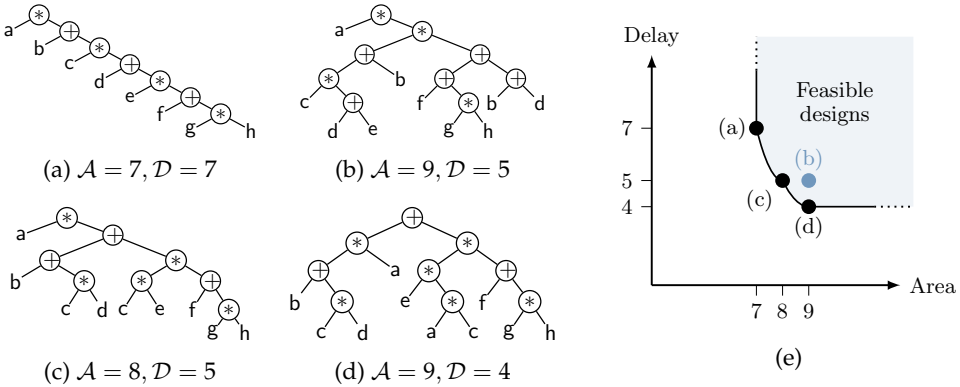
	I1	I2	N1	N2	N3	C1	O1
T = 0	✓	✓					
T = 1			✓	✓			
T = 2					✓	✓	
T = 3							✓

A visual illustration of *ACCNN*.

Node Type

```
PI   PO   CONSTANT   LUT 0x1   LUT 0x2   LUT 0x8
```

SeqEncoder: Optimization Sequence Feature Extraction

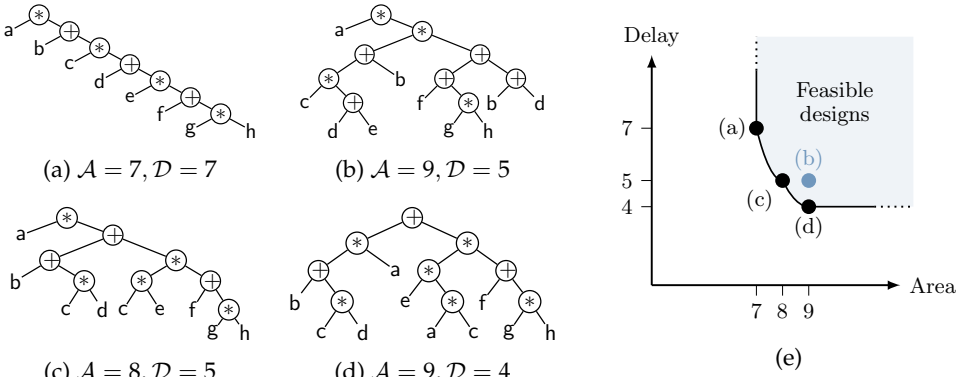


(a)–(d) Equivalent factored forms; (e) Area/delay trade-off for the trees.[14]

- Boolean expression $ab + acd + acef + acegh$
- Assume zero arrival time for all PIs, unit area ($\mathcal{A} = 1$), unit delay ($\mathcal{D} = 1$)

[14] Kanwar Jit Singh et al. (1988). "Timing optimization of combinational logic.". In: *Proc. ICCAD*.

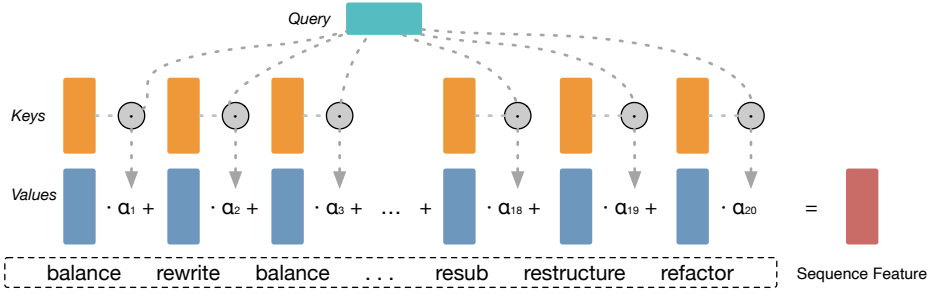
SeqEncoder: Optimization Sequence Feature Extraction



(a)–(d) Equivalent factored forms; (e) Area/delay trade-off for the trees.[14]

- It is hardly possible for designers to determine the effect of optimization sequences for different designs.
- We need a model that takes into account optimization sequence ordering and position.

[14] Kanwar Jit Singh et al. (1988). "Timing optimization of combinational logic.". In: *Proc. ICCAD*.

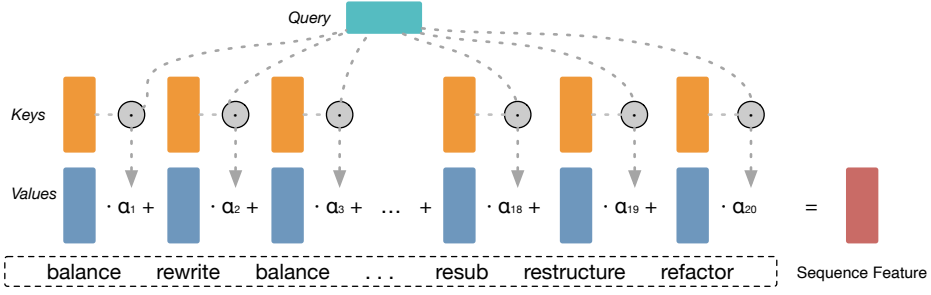


- Transformer[15] is one of such models

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}})\boldsymbol{V} \tag{3}$$

[15] Ashish Vaswani et al. (2017). "Attention is all you need". In: *Proc. NIPS*.

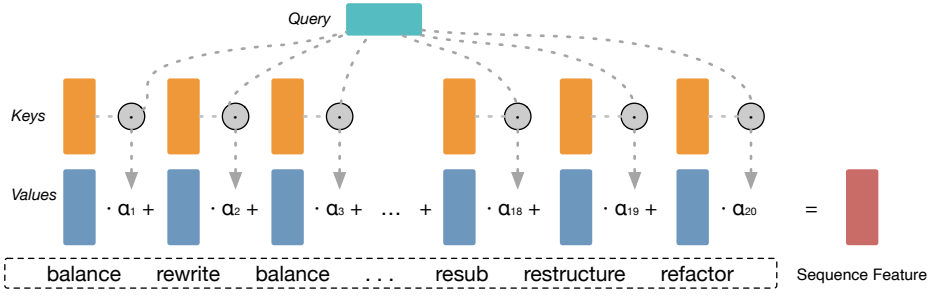
SeqEncoder: Optimization Sequence Feature Extraction



Optimization Methods

`Balancing, Reconfiguration, Replacing and Rewriting.`

SeqEncoder: Optimization Sequence Feature Extraction



SeqEncoder supports extracting features of optimization sequences of length 20 or less

When the length of the optimization sequence is less than 20

- Zero padding $\rightarrow$ *'empty optimization'*
- $[empty, rewrite, balance, \ldots, resub, restructure, refactor]$

Experiments

Experimental Setup

- Developed the timing prediction framework in Python
  - Tools: Yosys, ABC
  - Libraries: Pytorch Geometric, PyTorch, networkx
- Dataset: open-source designs

Type	IP	
	Train	Valid/Test
Bus protocol	i2c, spi ethernet, wb_dma simple_spi, pci wb_conmax	usb_phy ss_pcm sasc
Controller	ac97_ctrl, bp_be vga_lcd	mem_ctrl
Crypto	aes_secworks aes_xcrypt sha256	aes des3_area
DSP	fir, iir, jpeg	dft, idft
Processor	dynamic_node picosoc, tv80	fpu, tinyRocket

- Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{\hat{Y}_i - Y_i}{Y_i} \right| \tag{4}$$

Performance of LSTP

Table: Evaluation Accuracy (MAPE)

Name	# PI	# PO	# Node	# Level	SNS	Runtime [s]	LSTP	Runtime [s]
aes	683	529	39215	44	50.21%	2.85	**25.44%**	3.38
des3_area	303	64	7766	47	53.84%	2.16	**20.29%**	0.70
dft	37597	37417	488165	83	86.90%	27.18	**33.56%**	55.53
fpu	632	409	55935	1522	26.11%	4.96	**3.35%**	6.97
idft	37603	37419	481184	82	**5.07%**	16.16	8.18%	54.04
mem_ctrl	1187	962	29814	56	23.21%	32.02	**19.22%**	3.71
sasc	135	125	1214	15	21.44%	2.38	**2.48%**	0.12
ss_pcm	104	90	762	13	67.82%	2.30	**6.46%**	0.09
tinyRocket	4561	4181	99775	156	37.31%	81.87	**10.81%**	11.86
usb_phy	132	90	893	16	14.4%	2.65	**7.75%**	0.10
Average					38.63%	17.45	**13.75%**	**13.65**

- Our proposed method greatly outperforms prior works on all the test cases except for the `idft`.

LSTP for Optimization Sequence Generation

Table: Comparison of Timing minimums

Name	Initial [ns]	*resyn2*-2 [ns]	Improve [%]	LSTP [ns]	Improve [%]
aes	1.58	1.37	13.29%	1.24	**21.52%**
des3_area	2.66	3.74	-40.60%	3.33	-25.19%
dft	5.82	6.35	-9.11%	4.94	**15.12%**
fpu	51	41.5	18.63%	40.34	**20.90%**
idft	5.82	6.35	-9.11%	5.54	**4.81%**
mem_ctrl	6.74	3.03	55.04%	2.94	**56.38%**
sasc	0.89	0.69	22.47%	0.49	**44.94%**
ss_pcm	0.66	0.58	12.12%	0.48	**27.27%**
tinyRocket	78.1	12	84.64%	10.39	**86.70%**
usb_phy	0.41	0.42	-2.44%	0.32	**21.95%**
Average			14.49%		**27.44%**

- LSTP can be used to find a better optimization sequence

Conclusion

- Various tasks from architectural exploration to physical design DSE have highlighted the demand for fast logic synthesis result prediction
- In this paper, we proposed:
  - A machine learning driven logic synthesis timing predictor
  - A specialized GNN to sample and learn the intrinsic features of circuit AIG
  - An appropriate neural model to model the complex interaction between optimization passes and their effects on the input netlist
- We conducted comprehensive experiments on real-world circuit designs to evaluate our methods

THANK YOU!