A Highly Compressed Timing Macro-modeling Algorithm for Hierarchical and Incremental Timing Analysis

Tin-Yin Lai, and Martin D. F. Wong

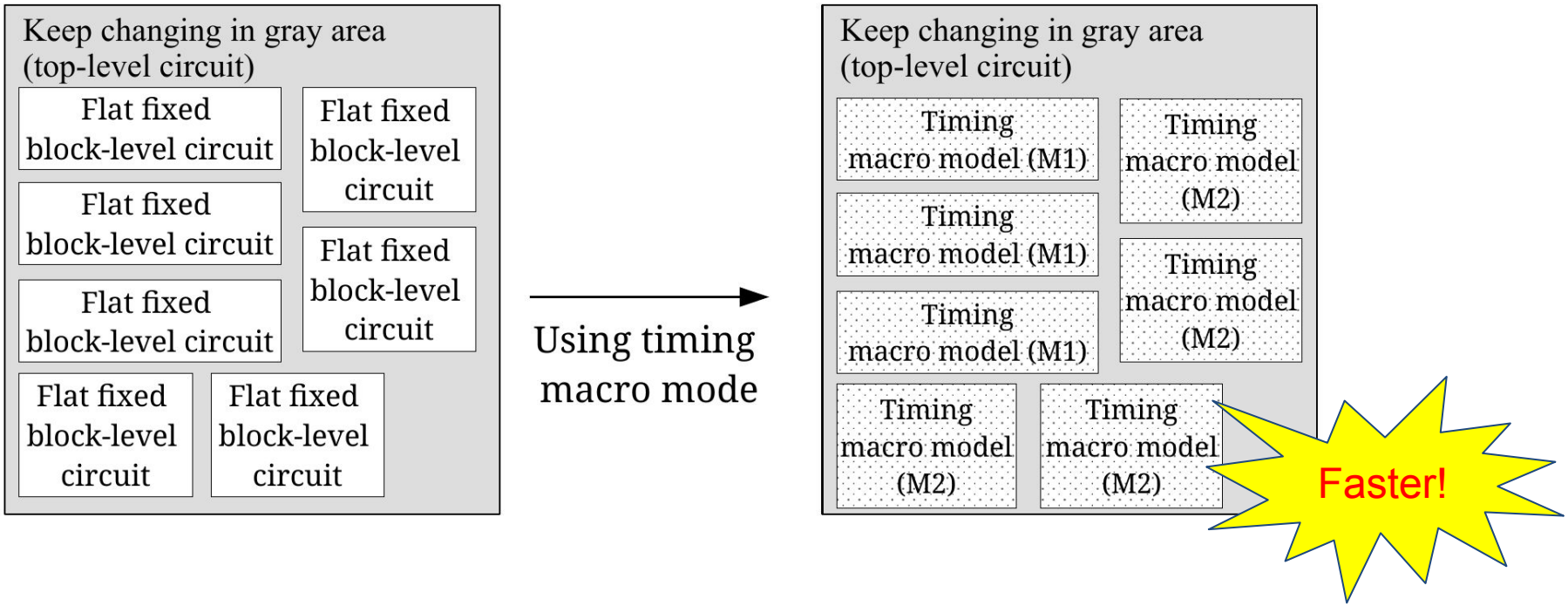
Jan. 23, 2018

Outline

- Introduction
  - Timing Macro-modeling
  - Problem Formulation
  - Previous Work - ILM
- Algorithm
  - Clocktree Construction
  - Forward Abs-tree (Out-tree) Graph Reduction
  - Cross Abs-edges Reduction
  - Constraint Reduction
  - Multi-threading using OpenMP
- Experimental Results
- Conclusion

Introduction

- Designs are large
  - Hierarchical timing analysis
  - Incremental timing analysis
- Highly compressed timing macro-models are needed

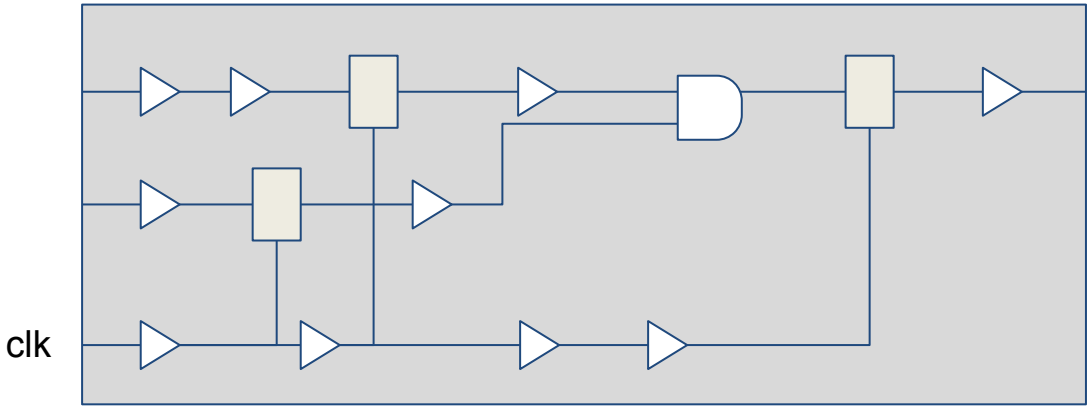


Problem Formulation

- Goal
  - **Accurate** boundary timing reproduction
  - **Small** model size
  - **Fast** runtime for timing analysis
  - **In-context usage** (incremental)
- Inputs
  - A set of circuit design
  - A set of boundary timing
    - Macro models usually are used under certain boundary timing
- Outputs
  - A timing macro model
    - The ability to reproduce timing information on primary input ports and primary output ports

Previous Work - Interface Logic Model (ILM)

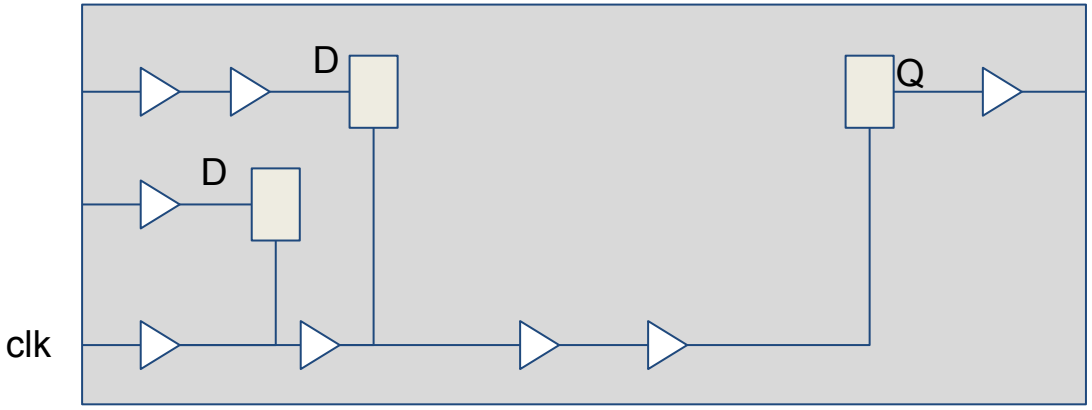
- The boundary timing for a block-level circuit is sufficient for timing macro usage in hierarchical timing analysis
  - Reproduce correct timing information on all the input ports and all the output ports
  - ILM keep nodes and edges that can only be observed from input ports and output port



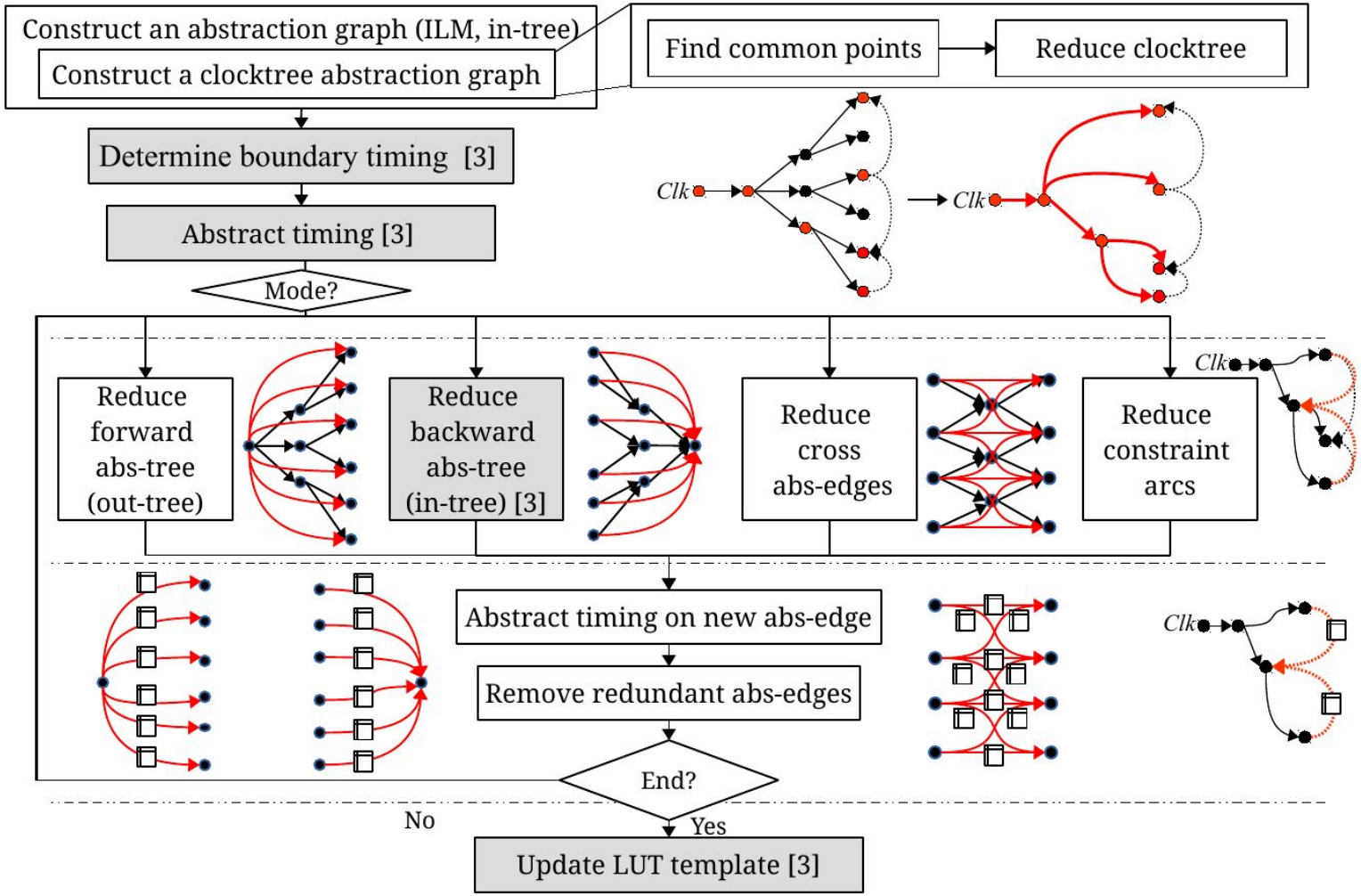
[1] A. J. Daga, L. Mize, S. Sripada, C. Wolff, and Q. Wu, “Automated timing model generation,” In Proc of DAC ’02.

Previous Work - Interface Logic Model (ILM)

- Implementation of ILM
  - Apply BFS from input ports until we find the first D pins
    - Back traverse to find all incoming timing paths for these D pins
  - Apply BFS from output ports back traverse until we find Q pins
  - We will deal with the clocktree later (keep it for now)



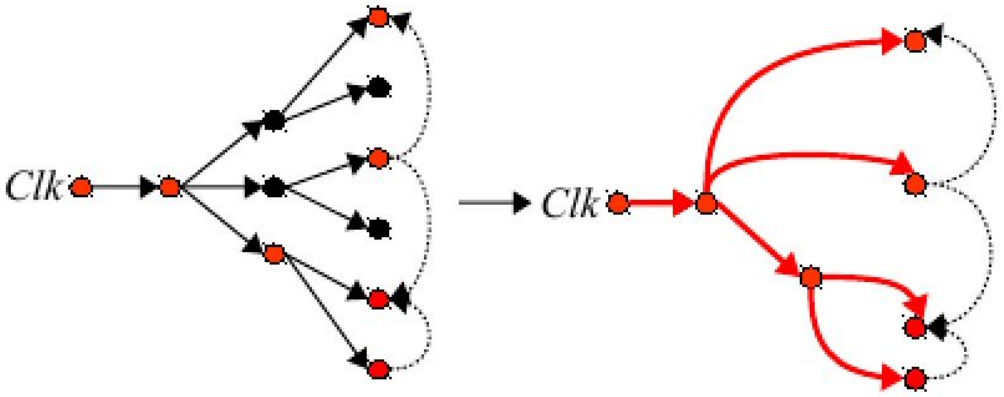
Algorithm - Program Flow



[3] T.-Y. Lai, T.-W. Huang, Martin D. F. Wong, "LibAbs: An Efficient and Accurate Timing Macro-Modeling Algorithm for Large Hierarchical Designs." Proceedings of the 54th ACM/IEEE Design Automation Conference - DAC '17 IEEE Press, 2017.

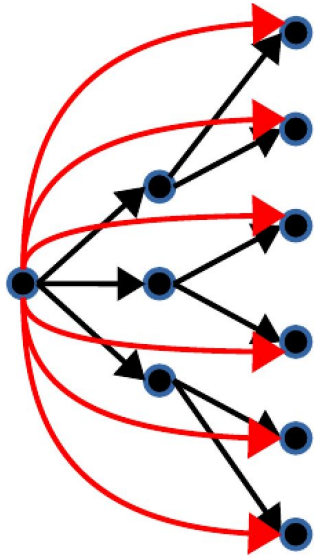
Algorithm - Clocktree reduction

- To maintain the CPPR
  - We have to keep the common point for any pairs of D pin and Q pin that exist timing paths
- Noted that there might be no timing path from the leaf pin of clocktree because ILM is applied
- Steps:
  - Find common points using dynamic programming
  - Construct the new clocktree from common points using BFS
    - Condition for BFS
      - Visited
      - Is common point
      - Is leaf of clocktree



Algorithm - Forward Abs-tree Graph Reduction

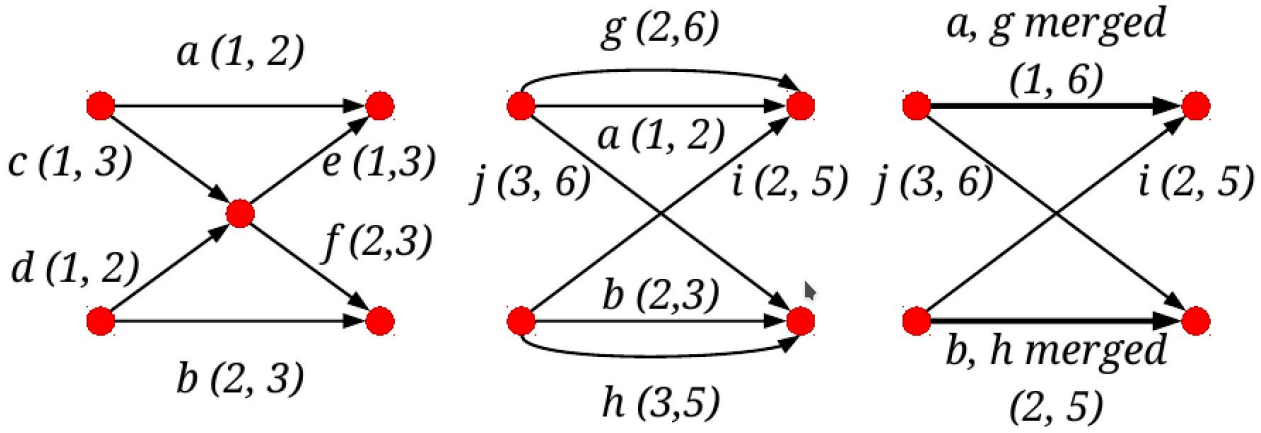
- Apply BFS to reduce forward tree structures
  - Condition for BFS in new timing graph construction
    - Multiple fanin edges
    - No fanout edges
    - Visited



Algorithm - Cross Abs-edges Reduction

- Cross structure
  - A node with multiple fanin edges and multiple fanout edges
- Connect from the fanin nodes to fanout nodes of the cross structure
  - Merge the new edges if there already exists a edge

- A 2-to-2 cross reduction example
  - Delay (min, max)

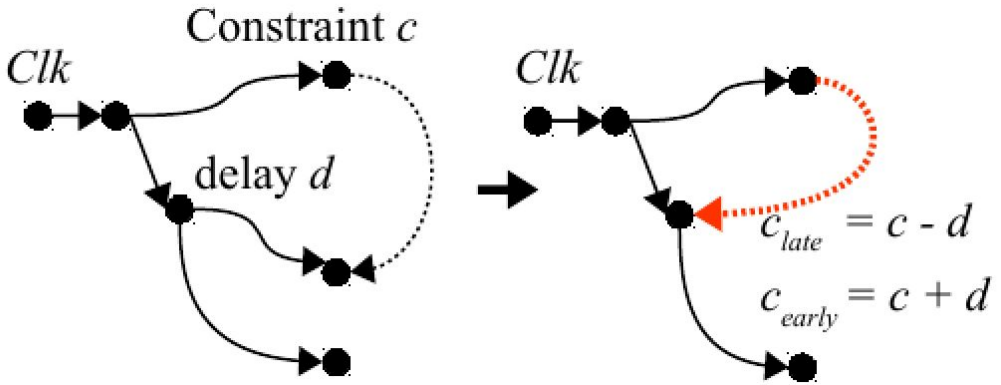


Algorithm - Constraint Reduction

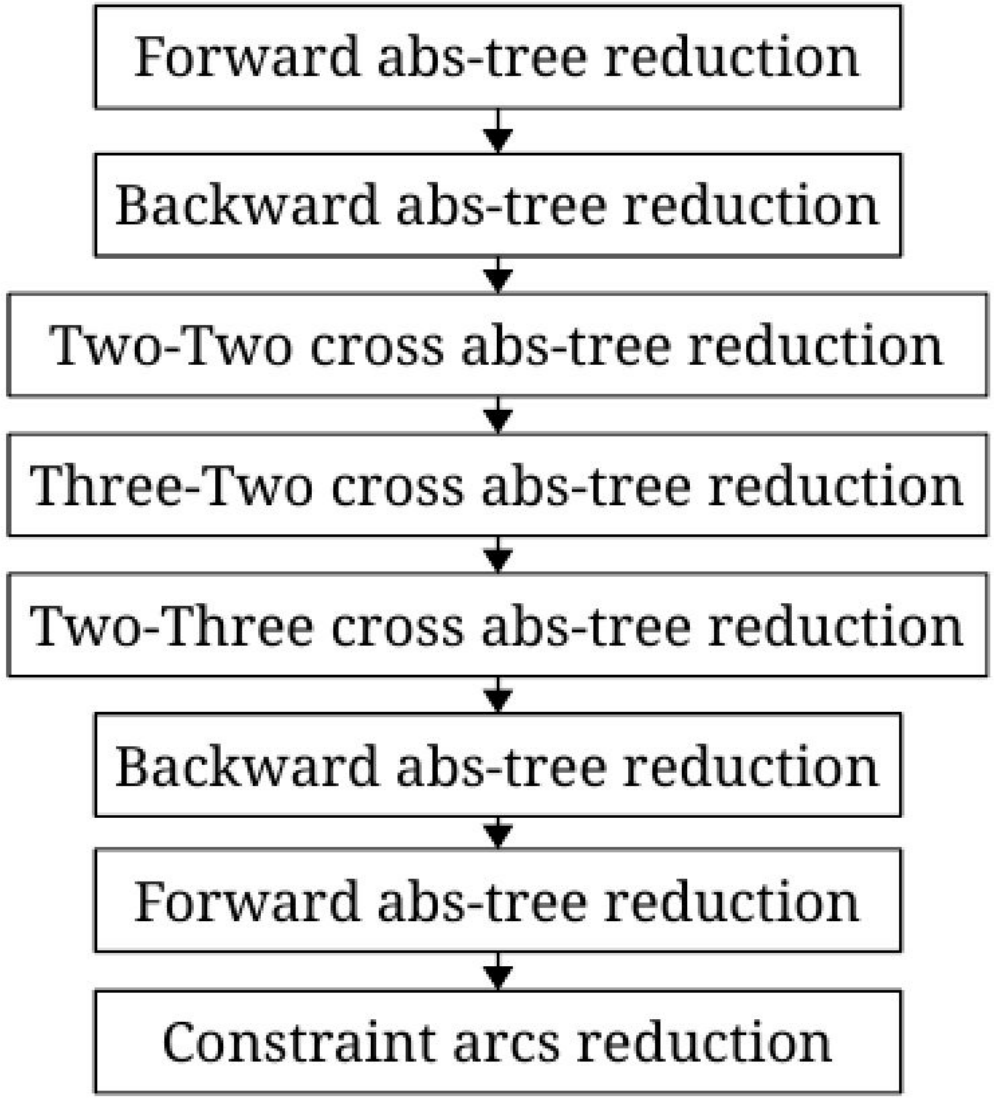
- Constraint edges provide timing constraints for calculating timing slacks
  - Include delay information on clocktree into constraint edges to reduce edges

$$constraint_{early} = constraint_{early}^{original} + dalay_{early}^{original} \quad (1)$$

$$constraint_{late} = constraint_{late}^{original} - dalay_{late}^{original} \quad (2)$$



Algorithm - Usage of Reduction Algorithms



Experimental Results (1)

- Accuracy, performance of macro-model generation

Circuits	Accuracy					Generate timing macro-model					
	Our work	LibAbs [3]		iTimerM [5]*		Our work		LibAbs [3]		iTimerM [5]*	
	max.	*max.*	*diff*	*max.*	*diff*	*runtime*	*mem*	*runtime*	*mem*	*runtime*	*mem*
mgc_edit_dist	0.15769	0.24414	0.08644	0.04	-0.11	15.018	<0.005	13.51	1.68	14.12	0.71
vga_lcd	0.25498	0.17749	-0.0775	0.03	-0.21	24.134	0.39	18.31	2.27	14.67	0.85
leon3mp	0.22044	0.25952	0.03908	0.04	-0.18	77.516	<0.005	109.64	11.85	54.65	4.05
netcard	0.20312	0.16796	-0.03516	0.06	-0.14	100.77	12.32	117.31	12.49	78.76	4.55
leon2	0.24101	0.16797	-0.07304	0.06	-0.18	104.56	14.94	136.66	14.97	113.32	5.59

- Macro usage (Non-incremental timing)

Circuits	Our work		LibAbs [3]				iTimerM [5]***			
	runtime	*mem*	*runtime*	*mem.*	$comp_{runtime}$	$comp_{mem}$	*runtime*	*mem.*	$comp_{runtime}$	$comp_{mem}$
mgc_edit_dist	4.345	<0.005	9.737	<0.005	44.62%	-	10.01	1.01	43.41%	<0.50%
vga_lcd	4.187	<0.005	11.151	<0.005	37.55%	-	9.44	0.99	44.35%	<0.51%
leon3mp	5.074	<0.005	64.983	12.00	7.81%	<0.04%	11.31	1.09	44.86%	<0.46%
netcard	22.524	4.23	68.855	12.54	32.71%	33.7%	47.42	5.12	47.50%	82.62%
leon2	38.417	5.19	81.799	15.361	46.97%	33.8%	74.94	8.17	51.26%	63.53%

[3] T.-Y. Lai, T.-W. Huang, Martin D. F. Wong, "LibAbs: An Efficient and Accurate Timing Macro-Modeling Algorithm for Large Hierarchical Designs." Proc of DAC '17
[5] P.-Y. Lee, Iris H.-R. Jiang, "iTimerM: Compact and Accurate Timing Macro Modeling for Efficient Hierarchical Timing Analysis." in Proc. of ISPD '17. ACM, 2017.

Experimental Results (2)

- Model size
  - Compared to [3]

Circuit	mgc_edit_dist				
Work	Original	LibAbs [3]		Our work	
Unit	Num	Num	%	Num	%
|N|	581319	95288	16.39%	46033	7.92%
|E|	691863	211461	30.56%	135234	19.55%
|Clk N|	67724	10449	15.43%	9753	14.40%
|Clk L|	153	39	25.49%	34	22.22%

Circui	vga_lcd					leon3mp				
Work	Original	LibAbs [3]		Our work		Original	LibAbs [3]		Our work	
Unit	Num	Num	%	Num	%	Num	Num	%	Num	%
|N|	768050	129240	16.83%	42791	5.57%	4167632	753406	18.08%	86582	2.08%
|E|	894826	273016	30.51%	119304	13.33%	4830700	1525362	31.58%	151689	3.14%
|Clk N|	205432	31555	15.36%	14208	6.92%	1305908	200905	15.38%	82042	6.28%
|Clk L|	191	56	29.32%	41	21.47%	245	76	31.02%	66	26.94%

Circuit	netcard					leon2				
Work	Original	LibAbs [3]		Our work		Original	LibAbs [3]		Our work	
Unit	Num	Num	%	Num	%	Num	Num	%	Num	%
|N|	4458141	783831	17.58%	282349	6.33%	5179094	949427	18.33%	449102	8.67%
|E|	5264603	1688054	32.06%	597399	11.35%	5974414	1894248	31.71%	1166964	19.53%
|Clk N|	1173710	180595	15.39%	142064	12.10%	1793548	275931	15.38%	158634	8.84%
|Clk L|	247	76	30.77%	68	27.53%	265	86	32.45%	72	27.17%

[3] T.-Y. Lai, T.-W. Huang, Martin D. F. Wong, "LibAbs: An Efficient and Accurate Timing Macro-Modeling Algorithm for Large Hierarchical Designs." Proc of DAC '17

Experimental Results (3)

- Model size
  - Compared to [5]
  - [5] reports their model size in file size

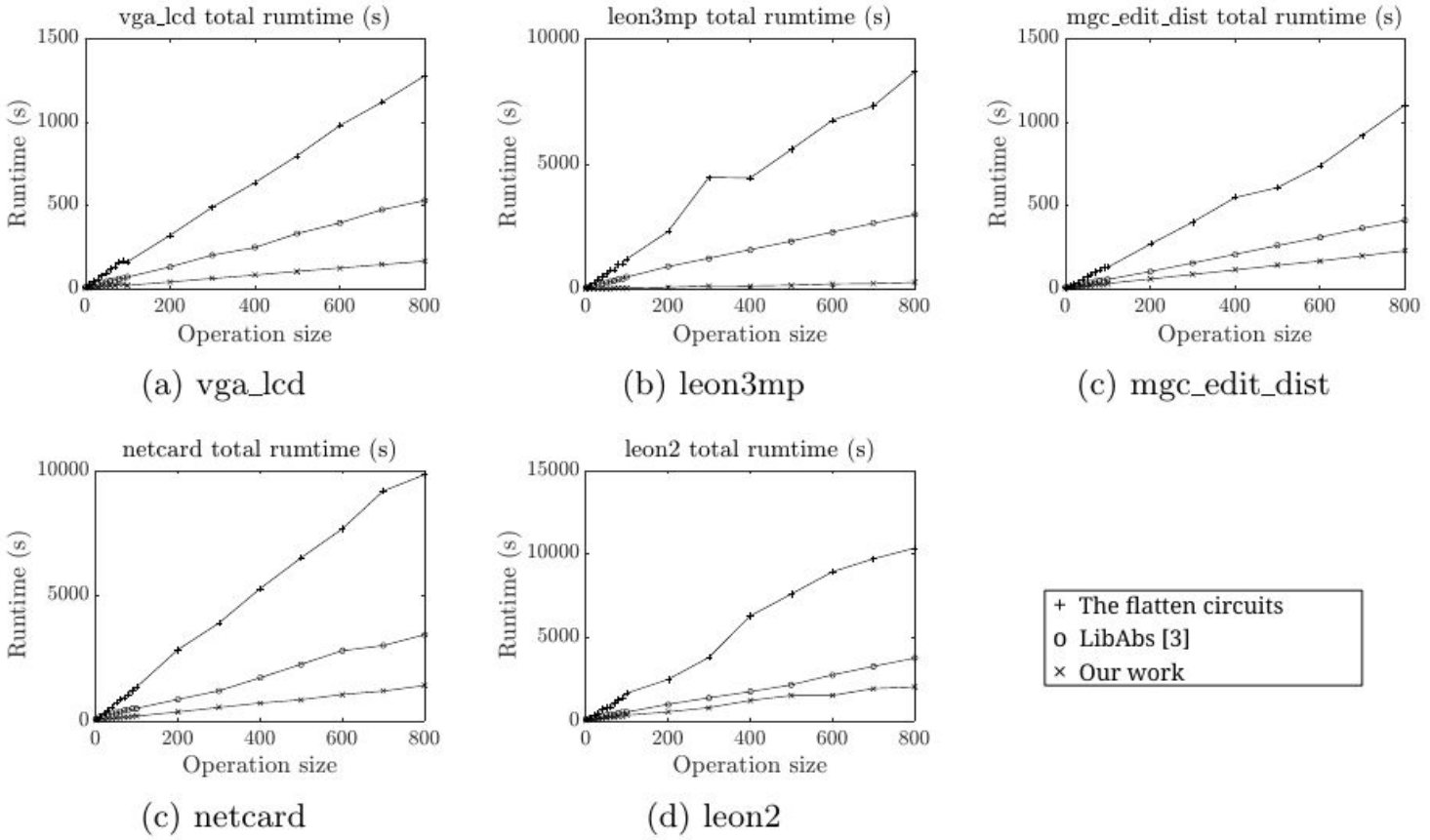
Circuit	mgc_edit_dist			vga_lcd			leon3mp		
Work	iTimerM [5]	Our work		iTimerM [5]	Our work		iTimerM [5]	Our work	
Unit	MB	MB	%	MB	MB	%	MB	MB	%
Liberty file size(MB)	90	79	87.7%	84	72	85.7%	96	86	89.6%

Circuit	netcard			leon2		
Work	iTimerM [5]	Our work		iTimerM [5]	Our work	
Unit	MB	MB	%	MB	MB	%
Liberty file size(MB)	435	372	85.5%	713	676	94.8%

[5] P.-Y. Lee, Iris H.-R. Jiang, “iTimerM: Compact and Accurate Timing Macro Modeling for Efficient Hierarchical Timing Analysis.” in Proc. of ISPD ’17. ACM, 2017.

Experimental Results (4)

- In-context usage (incremental timing)
  - x axis: # of incremental changes
  - y axis: runtime (s)



[3] T.-Y. Lai, T.-W. Huang, Martin D. F. Wong, "LibAbs: An Efficient and Accurate Timing Macro-Modeling Algorithm for Large Hierarchical Designs." Proc of DAC '17

Conclusions

- Our algorithm generates highly compressed timing macro-models efficiently
  - Accurate
    - About the same
  - Model size
    - Compared to the original timing graph
      - 9% in number of nodes
      - 19% in number of edges
  - Timing macro usage (non-incremental)
    - More than x2 times faster compared to the states of arts
  - In-context usage (incremental timing)
    - x5 times faster compared to the flat timing analysis
    - x1.7 times faster compared to the states of arts

Thank you!

Acknowledges

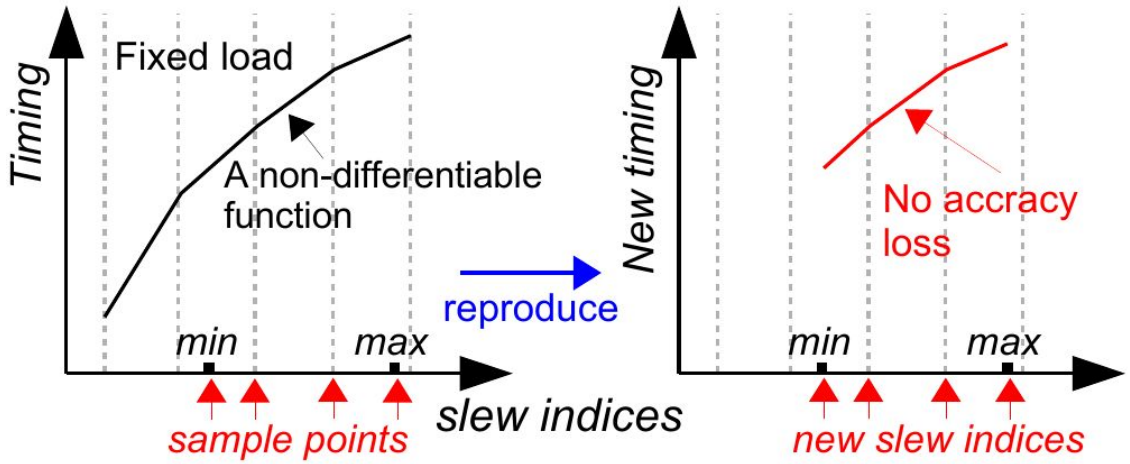
Prof. Martin Wong, UIUC CAD group, NCTU iTimerM, and 2017 TAU Timing Contest Committees

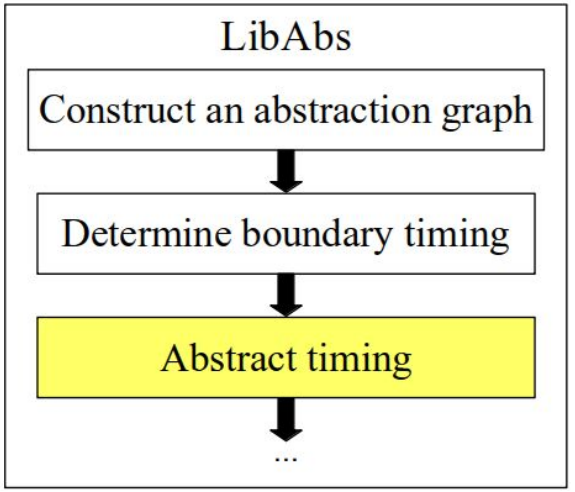
Timing Macro-modeling

- Timing macro-modeling
  - Abstracts timing behavior of a sub-design into a timing macro model to speed up the timing analysis
  - Speed up incremental optimization flow
    - In-context usage
  - An essential step in the hierarchical timing analysis

Algorithms - Abstract Timing - Initiate Indices (1)

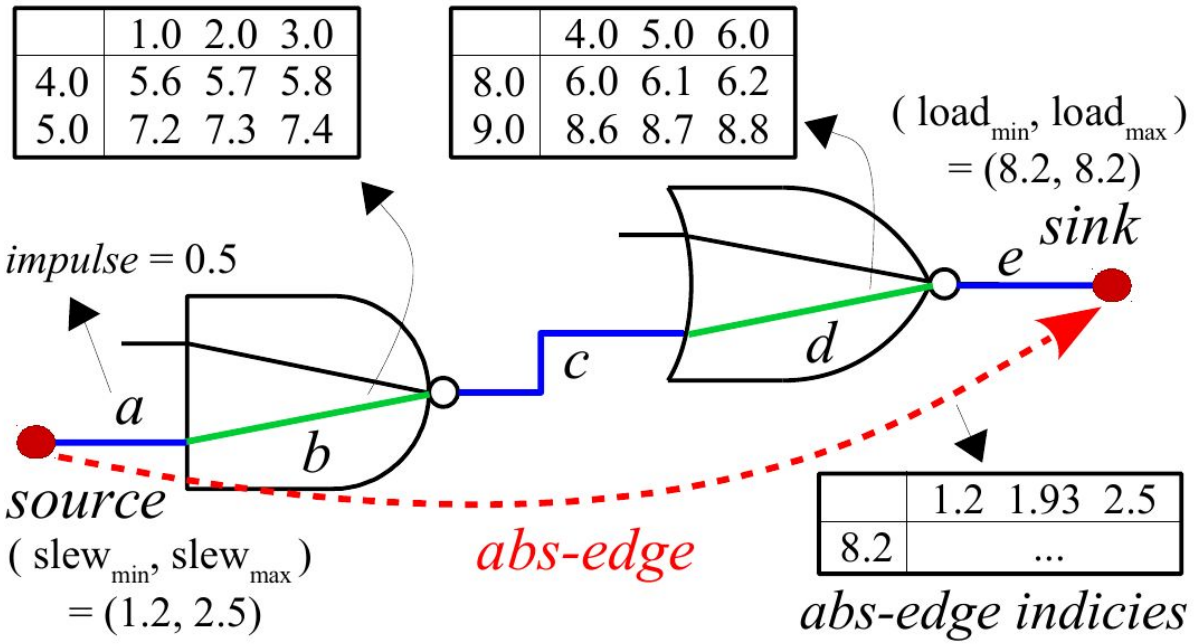
- Initiate indices
  - Delay and slew on wires
    - Based on the Elmore delay model
  - Delay and slew on cell arcs are non-differentiable functions
    - Derived from interpolation Look-Up Table
  - To minimize the accuracy loss
    - Sample on non-differentiable points

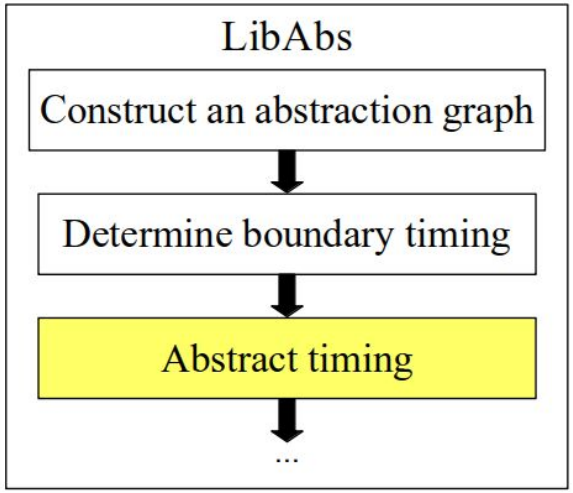




Algorithms - Abstract Timing - Initiate Indices (2)

- Initiate indices

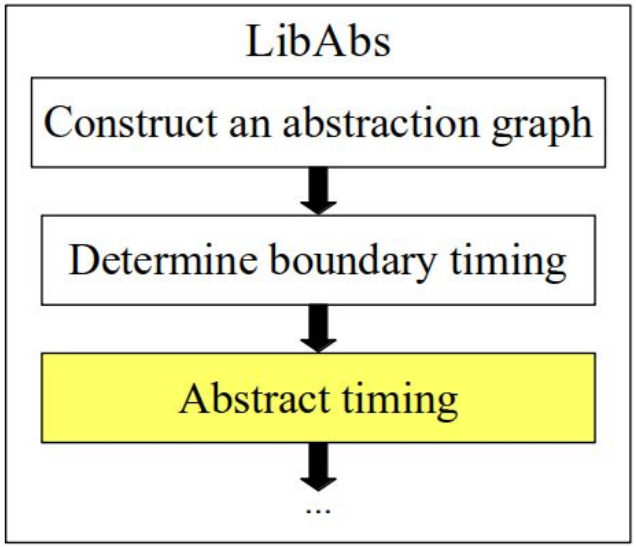




$$slew_{source} = \sqrt{slew_a^2 - impulse_a^2}$$

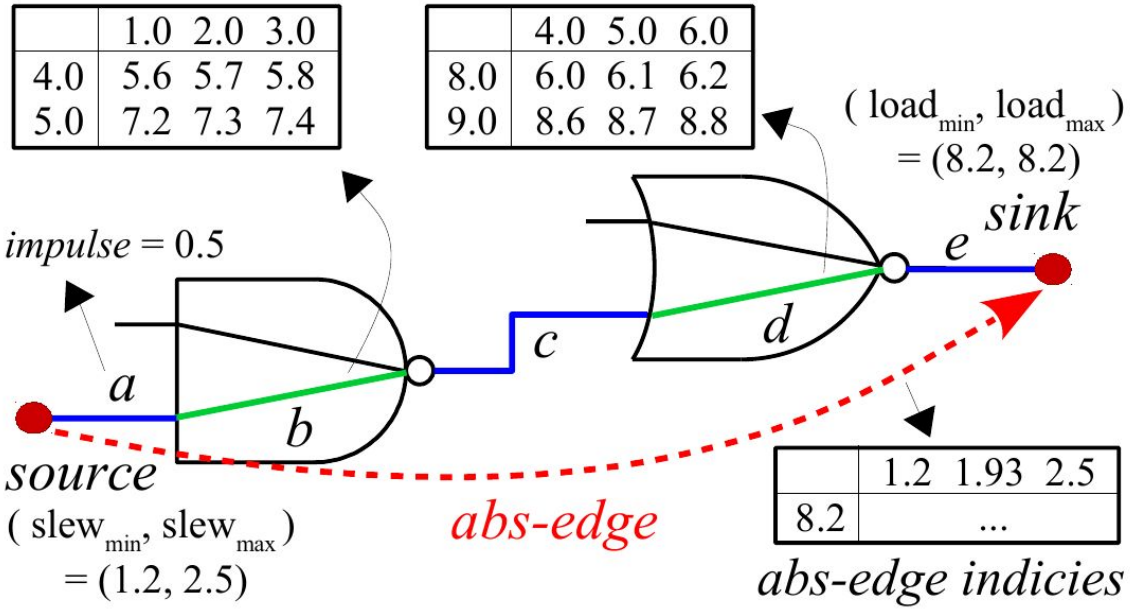
Algorithms - Abstract Timing - Infer Timing (3)

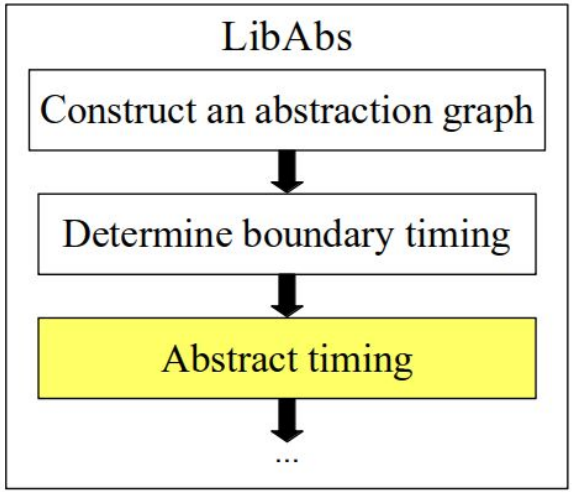
- Infer timing
  - Given a pair of (source slew, sink load)
  - $delay_{source-sink} = \sum delay\ values\ of\ corresponding\ edges$
  - $slew_{sink} = slew\ derived\ from\ LUT\ or\ parasitic\ wire$
    - LUT for cell arc
      - Interpolate the Look-Up Table
    - Wire parasitic
      - $slew_{sink} = \sqrt{slew_{source}^2 + impulse^2}$



Algorithms - Abstract Timing - Infer Timing (4)

- Infer timing





$$delay = delay_a + delay_b + delay_c + delay_d + delay_e$$

$$\begin{cases} slew_{sink} = LUT\ interpolation & b, d \\ slew_{sink} = \sqrt{slew_{source}^2 + impulse^2} & a, c, e \end{cases}$$