

# OpenTimer: A high-performance timing analysis tool

<https://github.com/OpenTimer/OpenTimer>

Tsung-Wei Huang and Martin D. F. Wong

Department of Electrical and Computer Engineering (ECE)

University of Illinois at Urbana-Champaign (UIUC), IL, USA



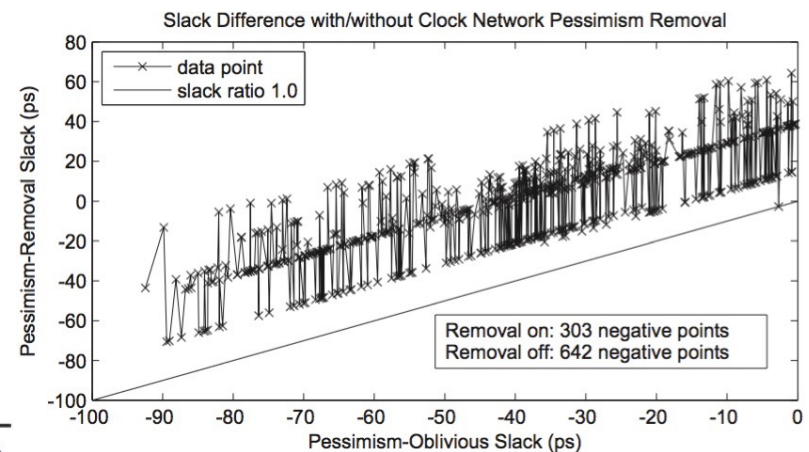
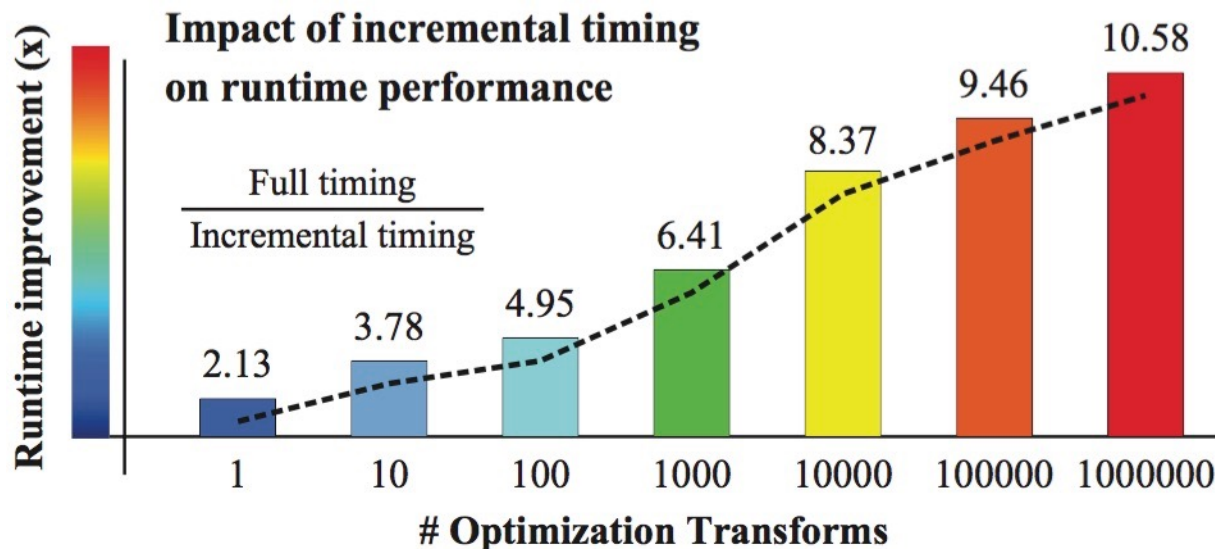
# Outline

---

- Problem formulation recap
  - Incremental timing and incremental CPPR
- OpenTimer architecture
  - Tool overview and features
  - Core algorithm (IO, graph reduction, pipeline scheduling)
- Experimental results
  - TAU 2015 contest benchmarks
- Conclusion

# Problem Formulation – TAU 2015 Contest

- A tool deals with incremental timing and incremental CPPR
  - Important for timing-driven applications
  - Fast full timing and incremental timing analysis
  - Capability of path-based CPPR analysis
  - Parallel programming and multi-threading

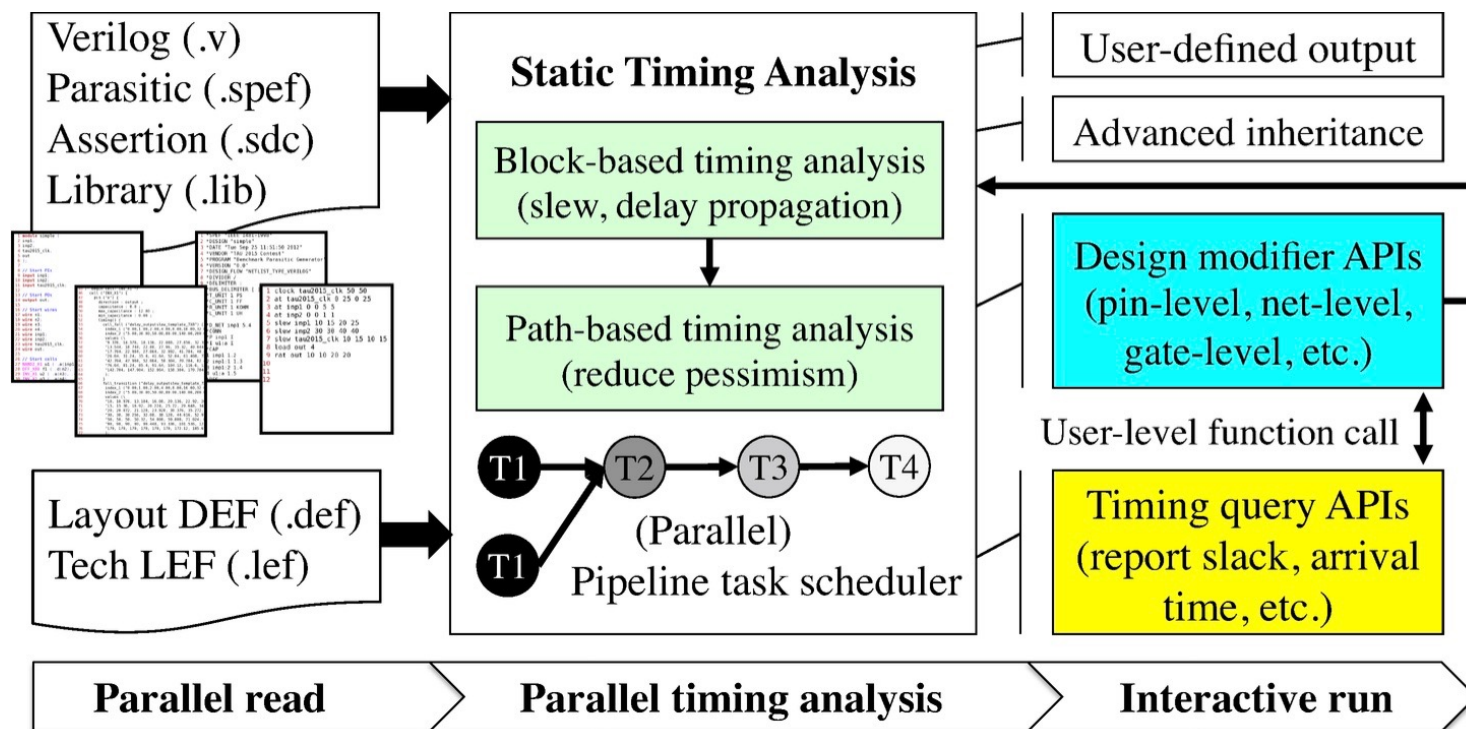


**CPPR impact**  
(reduction of unwanted pessimism)

*\*CPPR stands for Common Path Pessimism Removal*

# OpenTimer Architecture

- An open-source high-performance timing analysis tool
  - TAU14 (1<sup>st</sup> place), TAU15 (2<sup>nd</sup> place)
  - ICCAD15 (golden timer), TAU16 (golden timer)



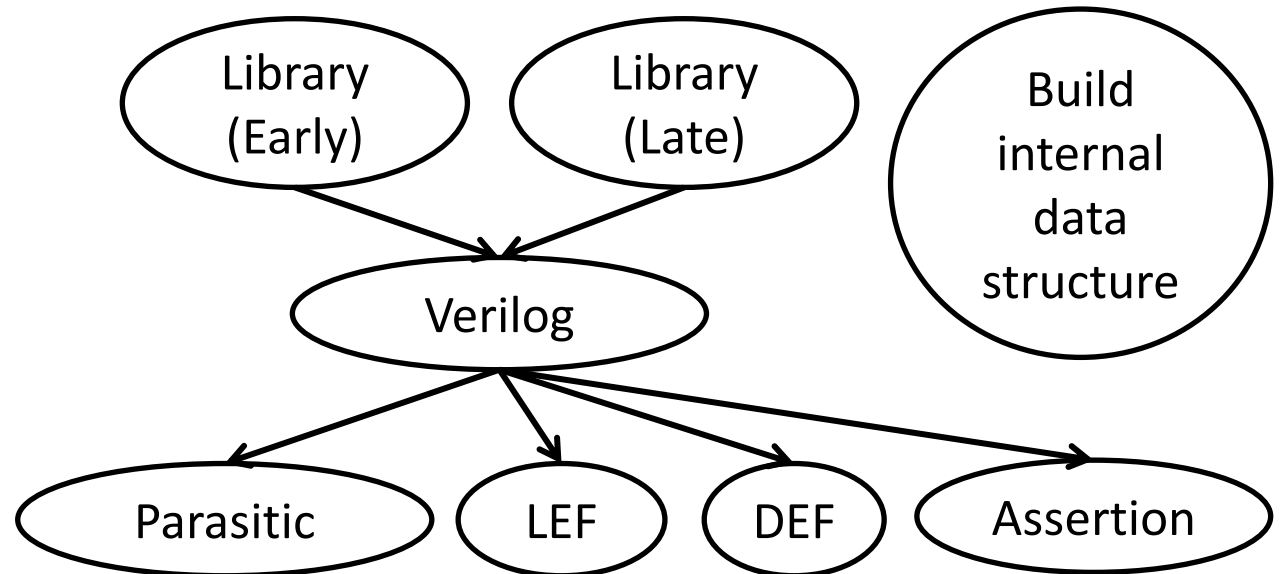
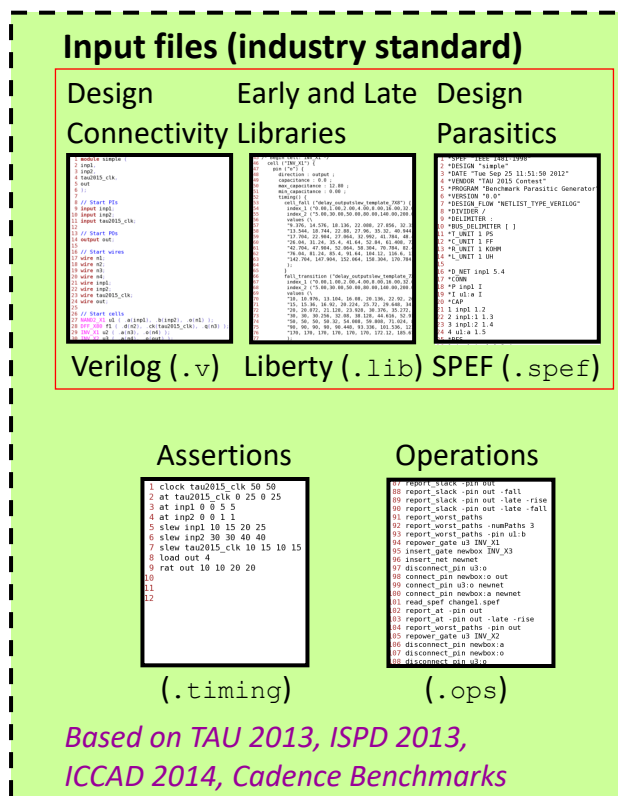
- Feature highlights
  - C++11
  - Industry format
  - STA engine
  - Block-based
  - Path-based
  - Incremental
  - Lazy evaluation
  - CPPR
  - Multi-threaded

<https://github.com/OpenTimer/OpenTimer>

# Initialize the Timer – Parallel IO

- A set of design files that follow the industry standard format
  - Verilog netlist, two libraries (early and late), parasitic spief, etc.
  - Time-consuming IO (e.g., file IO, complex parsing)

Task graph



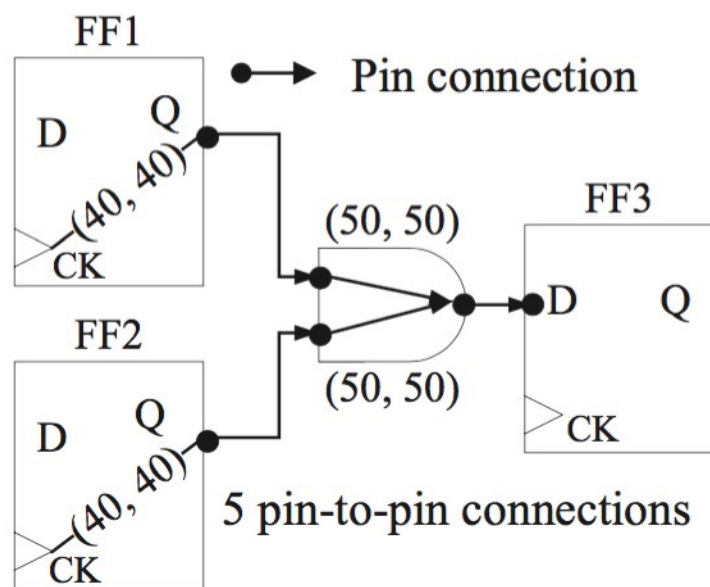
Parallel dependency generation using portable OpenMP  
 #pragma parallel ...  
 #pragma task ...

**x2** speedup by parallel read/parse!

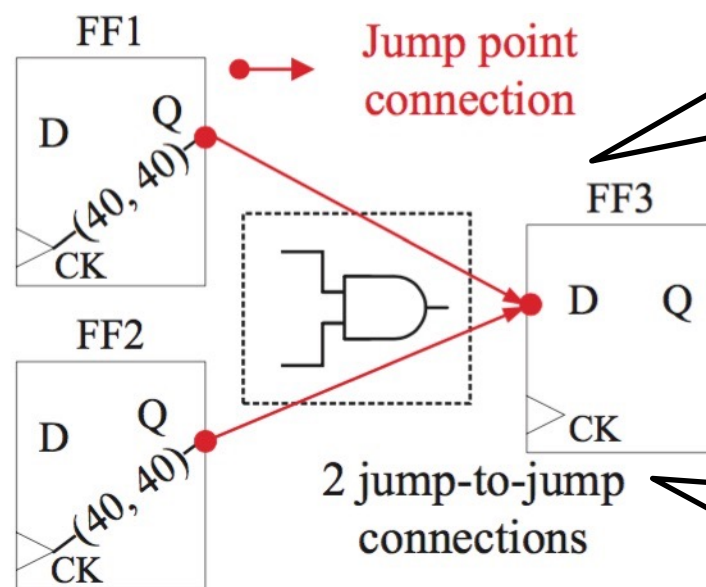


# Timing Graph Reduction

- Reduce the search space
  - Identify tree-structured subgraphs in the original timing graph
  - Merge every leaf-root path (transition-definite)



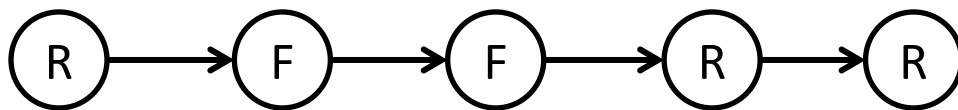
(a) Ordinary circuit graph



(b) Contracted circuit graph

~30% reduction  
on the graph  
size

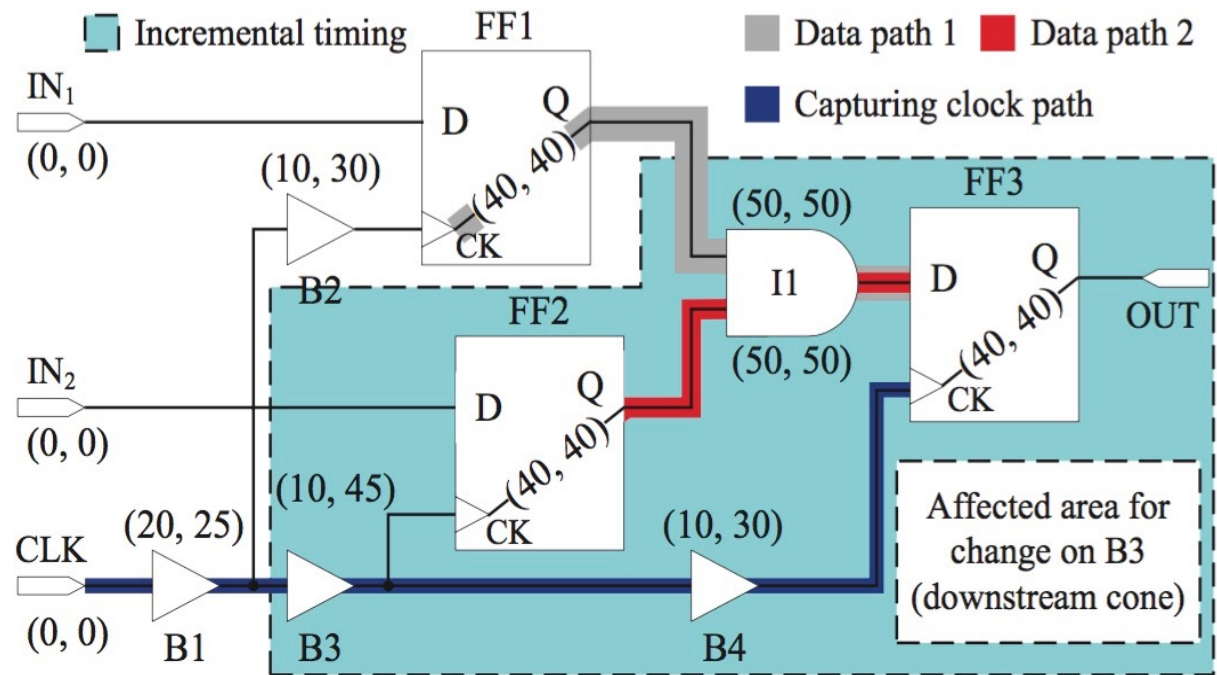
Apply to  
combinational  
circuits only



Every leaf-root path can be uniquely defined  
(given a transition at an endpoint)

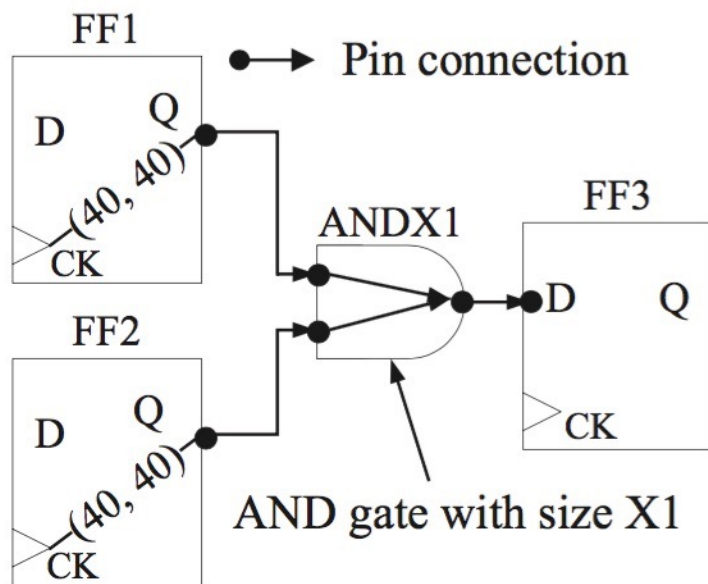
# Key Components of Incremental Timing

- Full timing is just a special case of incremental timing
- Design modifiers
  - Pin-level operations, net-level operations, and gate-level operations
- Timing queries
  - Slack
  - Arrival time
  - Required time
  - TNS and WNS
  - Critical path report
  - CPPR
- Source of propagation
- Lazy evaluation
- Explore parallel incremental timing

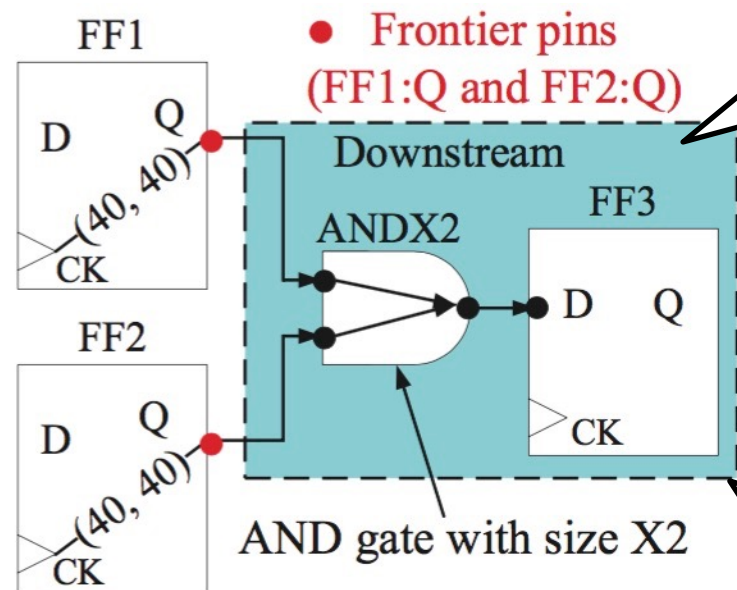


# Identify the Source of Incremental Timing (I)

- Source of incremental timing
  - Pins where the next timing propagation must originate from
  - Referred to as “**frontier pins**”
- Repower gate (gate sizing)
  - `repower_gate<gate_name, new_cell_name>`



(a) A circuit fragment

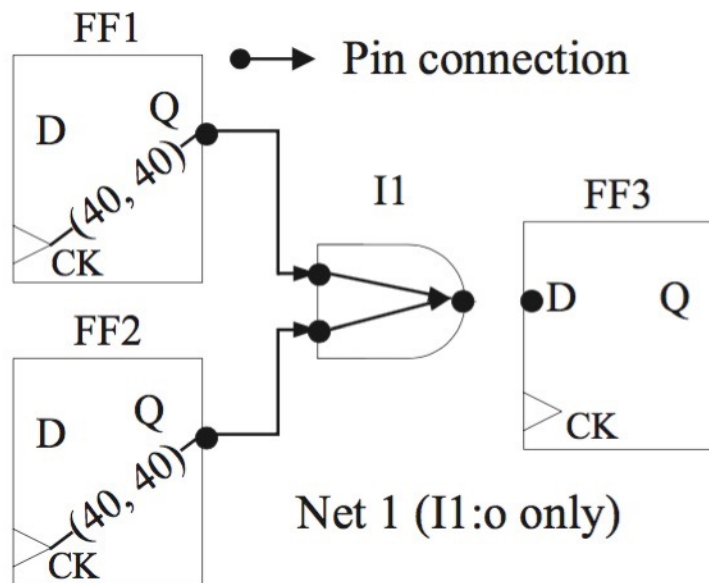


(b) Repower gate (X1→X2)

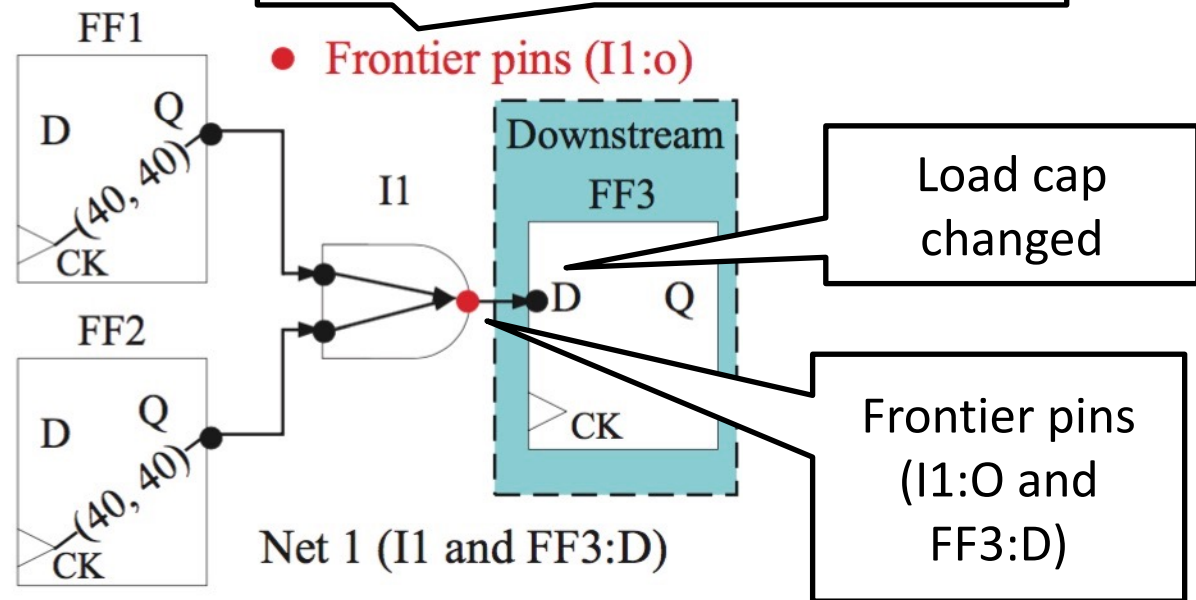


# Identify the Source of Incremental Timing (II)

- Disconnect pin (disconnect a pin from its net)
  - `disconnect_pin <pin_name>`
- Affect RC network only
  - Incremental timing from the root



(a) A circuit fragment

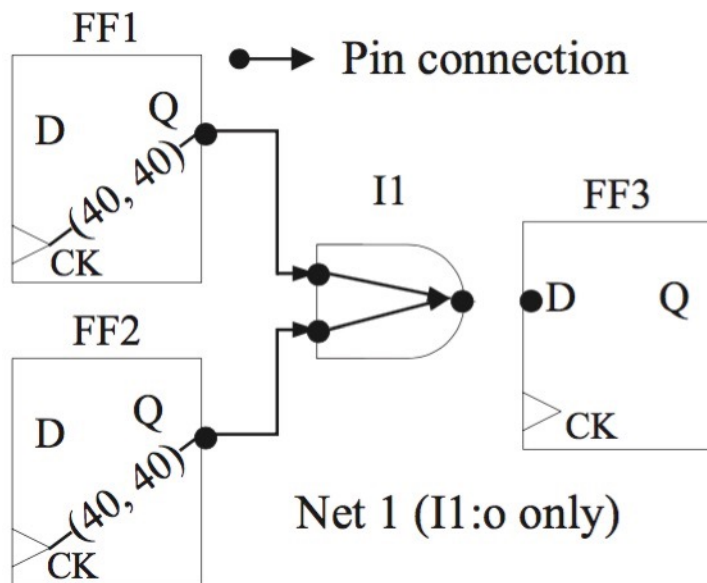


(b) Connect pin FF3:D to net 1

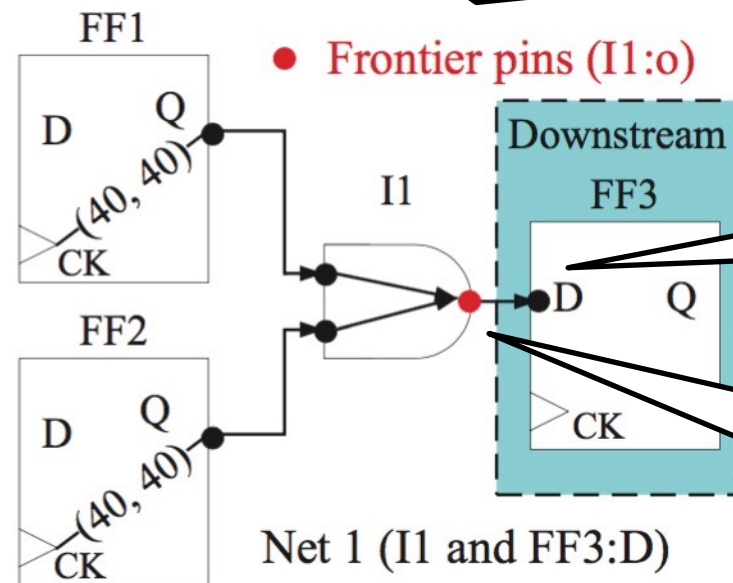
# Identify the Source of Incremental Timing (III)

- Connect pin (connect a pin to a net)
  - connect\_pin <pin\_name, net\_name>
- Affect RC network only
  - Incremental timing from the root

Find the root of the corresponding RC network (RC tree)



(a) A circuit fragment



(b) Connect pin FF3:D to net 1

Load cap changed

Frontier pins (I1:O and FF3:D)

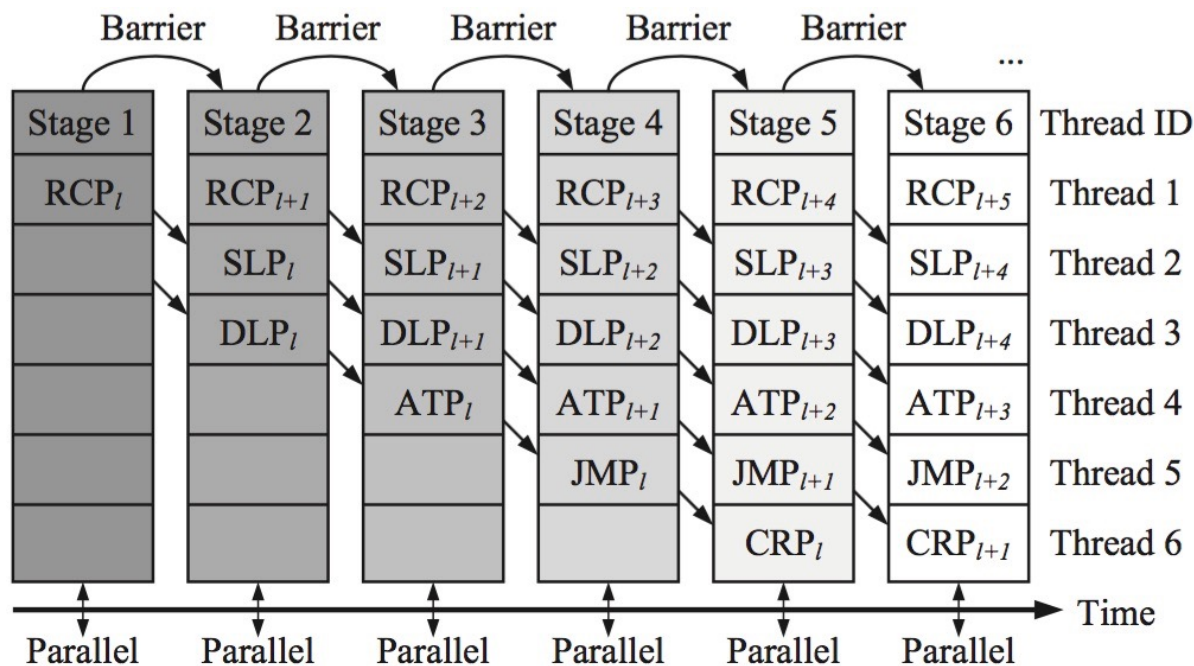
# Dealing with Design Modifiers

---

- Design modifiers can be built upon pin-level modification
  - remove\_net: disconnect all pins and remove the net
  - insert\_net: create an empty net
  - insert\_gate: insert a gate and create pins
  - remove\_gate: disconnect all input and output pins, and remove the gate
  - ...
- How to store frontier pins for correct timing propagation?
  - Timing graph is a directed acyclic graph (DAG)
  - Efficient data structure, *bucketlist*, to maintain the dependency
    - Apply topological sort (longest path finding)
    - Level index of each pin to keep track of dependency
    - Every frontier pin is inserted into the corresponding bucket
    - Incremental topological sort for incremental levelization

# Pipeline-based Parallel Timing Propagation

- Timing propagation has several linearly dependent tasks
  - RC update  $\rightarrow$  Slew & Delay  $\rightarrow$  Arrival time  $\rightarrow$  Jump point  $\rightarrow$  CPPR
  - Pipeline scheduling with multiple threads



We use the following paper for dealing with CPPR

\*UI-Timer: An ultra-fast clock network pessimism removal algorithm,  
T.-W. Huang, P.-C. Wu, and Martin D. F. Wong, ICCAD14

Algorithm 18: update\_timing()

```

1   $B \leftarrow$  bucket list of the timer;
2  if  $B.num\_pins = 0$  then
3    return;
4  end
5  IncrementalLevelization( $B$ );
6   $l_{min} \leftarrow B.min\_nonempty\_level$ ;
7   $l_{max} \leftarrow B.max\_nonempty\_level$ ;
8  # Parallel_Region {
9  # Master_Thread_do for  $l = l_{min}$  to  $l_{max} + 4$  do
10   # Fork_Thread_Task PropagateRC( $l$ );
11   # Fork_Thread_Task PropagateSlew( $l - 1$ );
12   # Fork_Thread_Task PropagateDelay( $l - 1$ );
13   # Fork_Thread_Task PropagateArrivalTime( $l - 2$ );
14   # Fork_Thread_Task PropagateJumpPoint( $l - 3$ );
15   # Fork_Thread_Task PropagateCPPRCredit( $l - 4$ );
16   # Synchronize_Thread_Tasks;
17 end
18 };
19 # Parallel_Region {
20 # Master_Thread_do for  $l = l_{max}$  to  $B.min\_non\_empty\_level$  do
21   # Fork_Thread_Task PropagateFanin( $l$ );
22   # Fork_Thread_Task PropagateRequiredArrivalTime( $l$ );
23   # Synchronize_Thread_Tasks;
24 end
25 };
26 remove all pins from the bucket list  $B$ ;
    
```

# Experimental Results – Environment Setup

- Implementation
  - C++11 with GCC 4.8
  - Linux machine (8 cores)\*
- Benchmark suite
  - TAU15 contest benchmarks
- Baseline on TAU15 winners
  - iTimerC 2.0 (1<sup>st</sup> place)
  - iitRACE (3<sup>rd</sup> place)

Design	Number of:			
	PIs	POs	Gates	Nets
vga_lcd	89	109	139.5K	139.6K
cordic_ispd	34	64	45.4K	45.4K
des_perf_ispd	234	140	138.9K	139.1K
edit_dist_ispd	2.6K	12	147.6K	150.2K
fft_ispd	1.0K	2.0K	38.2K	39.2K
* cordic_dut	80	78	3.6K	3.6K
* crc32d16N	19	32	478	495
* softusb_navre	34	51	6.9K	7.0K
* tip_master	778	869	37.7K	38.5K
b19_iccad	22	25	255.3K	255.3K
mgc_edit_dist_iccad	2.6K	12	161.7K	164.2K
mgc_matrix_mult_iccad	3.2K	1.6K	171.3K	174.5K
vga_lcd_iccad	85	99	259.1K	259.1K
netcard_iccad	1.8K	10	1496.0K	1497.8K
leon2_iccad	615	85	1616.4K	1517.0K
leon3mp_iccad	254	79	1247.7K	1248.0K

*\*clock tree has inverters and buffers*

**7** based on released benchmarks ( $\sim 10^2 - \sim 10^5$  gates)

**3** based on Cadence benchmarks ( $\sim 10^2 - \sim 10^5$  gates)

**6** based on ICCAD 2014 benchmarks ( $\sim 10^5 - \sim 10^6$  gates)

*\*Campus cluster, University of Illinois at Urbana-Champaign (UIUC)*

<https://campuscluster.illinois.edu/>



# Experimental Results – Overall Performance Comparison

TABLE I

PERFORMANCE COMPARISON BETWEEN OPENTIMER AND TOP-RANKED TIMERS iitRACE AND iTimerC 2.0 FROM TAU 2015 CAD CONTEST [1].

Circuit	#Gates	#Nets	#OPs	iitRACE			iTimerC 2.0			OpenTimer		
				accuracy	runtime	memory	accuracy	runtime	memory	accuracy	runtime	memory
b19	255.3K	255.3K	5641.5K	63.03 %	629 s	3.0 GB	99.95 %	215 s	5.8 GB	99.95 %	52 s	4.6 GB
cordic	45.4K	45.4K	1607.6K	61.83 %	100 s	0.9 GB	98.88 %	80 s	1.3 GB	98.88 %	18 s	1.3 GB
des_perf	138.9K	139.1K	4326.7K	67.43 %	299 s	4.2 GB	97.02 %	92 s	3.1 GB	99.73 %	30 s	3.0 GB
edit_dist	147.6K	150.2K	3368.3K	64.83 %	857 s	2.0 GB	98.29 %	98 s	3.8 GB	98.30 %	42 s	3.8 GB
fft	38.2K	39.2K	1751.7K	89.66 %	70 s	0.5 GB	98.45 %	49 s	1.2 GB	99.77 %	11 s	1.2 GB
leon2	1616.4K	1517.0K	8438.5K	72.34 %	16832 s	9.9 GB	100.00 %	787 s	27.2 GB	100.00 %	282 s	22.8 GB
leon3mp	1247.7K	1248.0K	8405.9K	62.99 %	4960 s	8.2 GB	100.00 %	609 s	19.8 GB	100.00 %	163 s	17.9 GB
mgc_edit_dist	161.7K	164.2K	3403.4K	64.29 %	1578 s	1.9 GB	100.00 %	135 s	4.1 GB	100.00 %	41 s	3.1 GB
mgc_matrix_mult	171.3K	174.5K	3717.5K	67.93 %	1363 s	2.0 GB	100.00 %	157 s	4.3 GB	100.00 %	31 s	3.1 GB
netcard	1496.0K	1497.8K	11594.6K	87.63 %	6662 s	9.4 GB	99.99 %	691 s	22.9 GB	99.99 %	192 s	20.8 GB
cordic_core	3.6K	3.6K	226.0K	59.42 %	21 s	0.3 GB	95.19 %	29 s	0.2 GB	95.19 %	3 s	0.1 GB
crc32d16N	478	495	28.9K	57.15 %	3 s	0.1 GB	100.00 %	5 s	0.1 GB	100.00 %	1 s	0.1 GB
softusb_navre	6.9K	7.0K	427.8K	40.17 %	21 s	0.1 GB	0.00 %	-	-	99.97 %	4 s	0.5 GB
tip_master	37.7K	38.5K	1300.4K	82.95 %	64 s	0.6 GB	96.42 %	47 s	1.0 GB	97.04 %	9 s	0.8 GB
vga_lcd_1	139.5K	139.6K	2961.5K	99.65 %	260 s	1.6 GB	100.00 %	94 s	2.2 GB	100.00 %	31 s	2.9 GB
vga_lcd_2	259.1K	259.1K	12674.7K	98.57 %	1132 s	13.3 GB	100.00 %	156 s	5.0 GB	100.00 %	65 s	3.9 GB

#Gates: number of gates. #Nets: number of nets. #OPs: number of operations. accuracy: average of path accuracy and value accuracy (%). -: program crash.

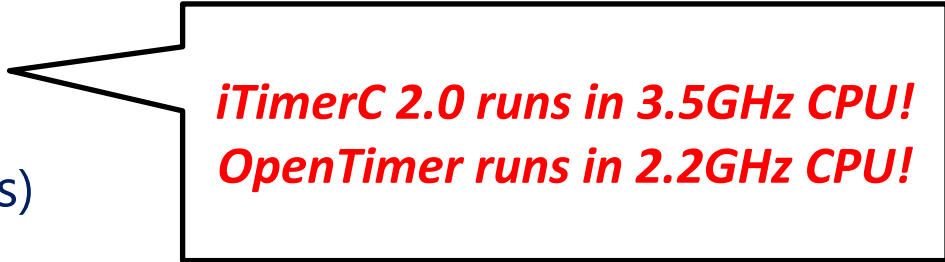
*\*iTimerC 2.0: 1<sup>st</sup> place in TAU contest 2015 (binary from authors)*

*\*iitRACE: 3<sup>rd</sup> place in TAU contest 2015 (binary from authors)*

# Experimental Results – Performance Highlights

---

- OpenTimer performance highlights
  - Achieved the highest accuracy (both path-based and value-based)
  - Achieved the fastest runtime (x2 to x9 speedup)
  - Robust and reliable (no crash)
  - Small memory usage\*
- Comparison to the *newest* results of iTimerC 2.0\*\*
  - netcard (1.5M gates and 1.5M nets)
    - OpenTimer: 192s, 20GB
    - iTimerC 2.0: 213s, 21GB
  - leon3mp (1.2M gates and 1.2M nets)
    - OpenTimer: 163s, 18GB
    - iTimerC 2.0: 186s, 19GB



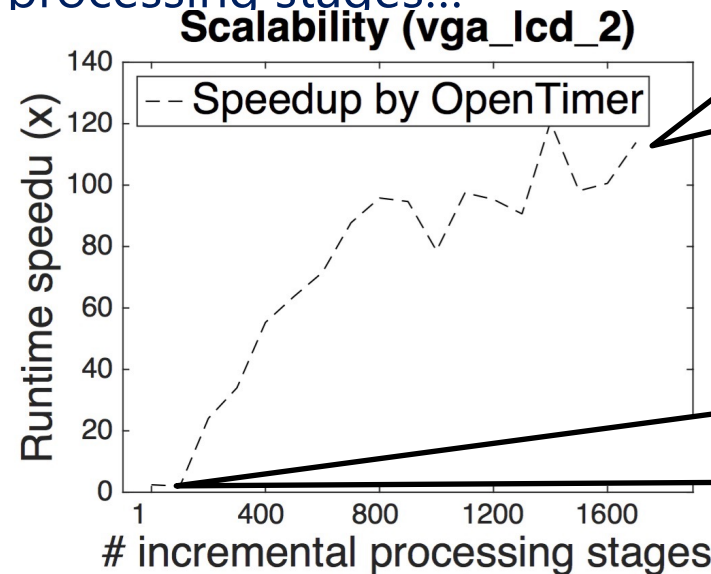
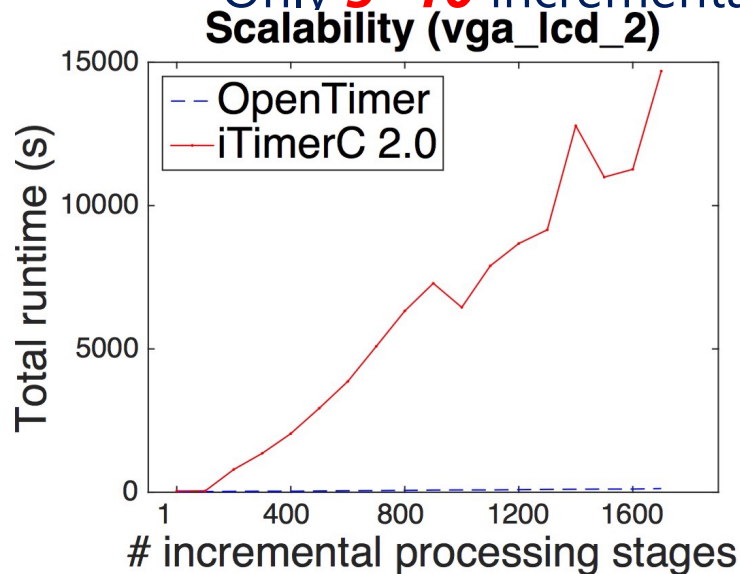
***iTimerC 2.0 runs in 3.5GHz CPU!***  
***OpenTimer runs in 2.2GHz CPU!***

*\* Our contest version has higher memory usage is due to the different system environment settings between IBM machine and UIUC machine*

*\*\*iTimerC 2.0: Fast incremental timing and CPPR analysis, ICCAD15*

# Experimental Results – Scalability of Incremental Timing

- Optimization or synthesis tools
  - Call an incremental timer millions of times to optimize the timing objective
- One incremental processing stage
  - A set of design modifiers followed by a timing query
- Insufficiency of TAU15 benchmarks
  - Only **5~10** incremental processing stages...

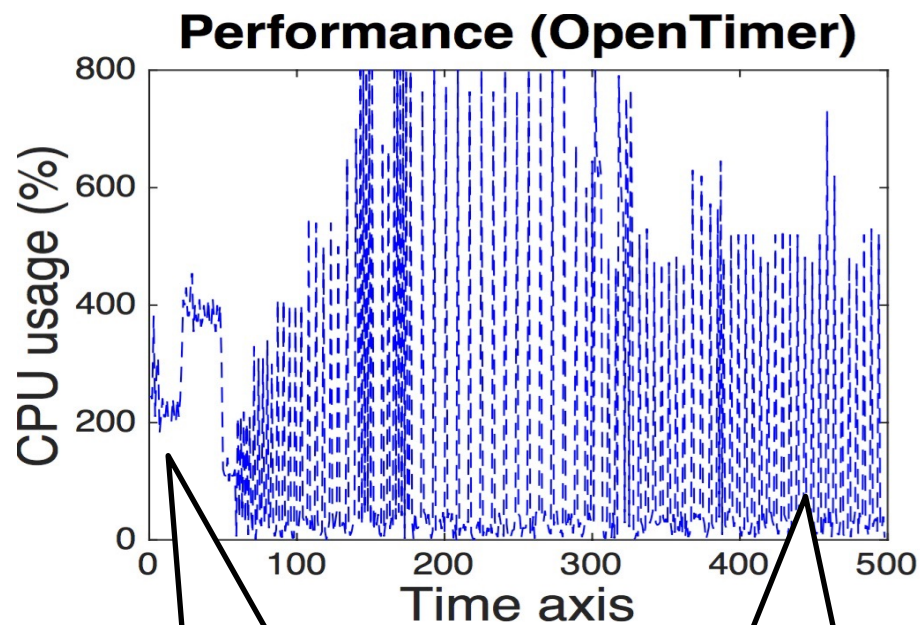


**x115 speedup** by  
OpenTimer  
(at 1459th stage)

**x2.7 speedup** by  
OpenTimer  
(at 1th stage, i.e.  
full timing)

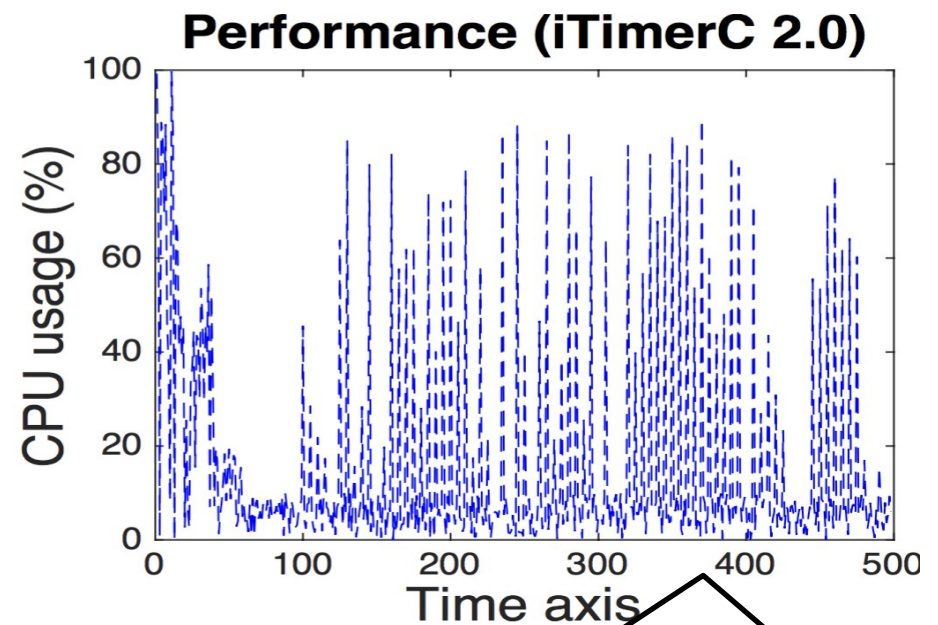
# Experimental Results – Parallelism Comparison

- Capability of multi-threading
  - Higher parallelism translates to higher cpu usage
  - Better resource utilization reflects on higher cpu usage



**Parallel IO** by  
OpenTimer

**Parallel timing**  
by OpenTimer



**Single thread** by iTimerC 2.0  
(no parallelism)



# Conclusion

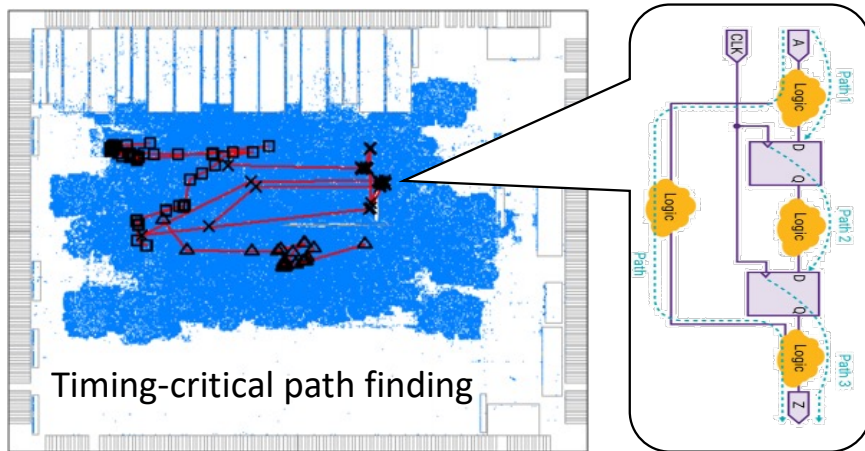
---

- Developed a high-performance timing analysis tool
  - Free software and open-source under GPL v3.0
  - Industry format (.v, .spdef, .lib, .lef, .def, etc.)
  - Fast, accurate, and robust
  - Multi-threaded and CPPR by default
- Recognition
  - 1<sup>st</sup> prize in TAU14 contest (full timing with CPPR)
  - 2<sup>nd</sup> prize in TAU15 contest (incremental timing with CPPR)
  - Golden timer in ICCAD15 CAD contest
  - Golden timer in TAU16 contest
- Acknowledgment
  - Jin, Billy, M.-C., team iTimerC, team iitRACE, and the UIUC CAD group!



# OpenTimer Community

<https://github.com/OpenTimer/OpenTimer>



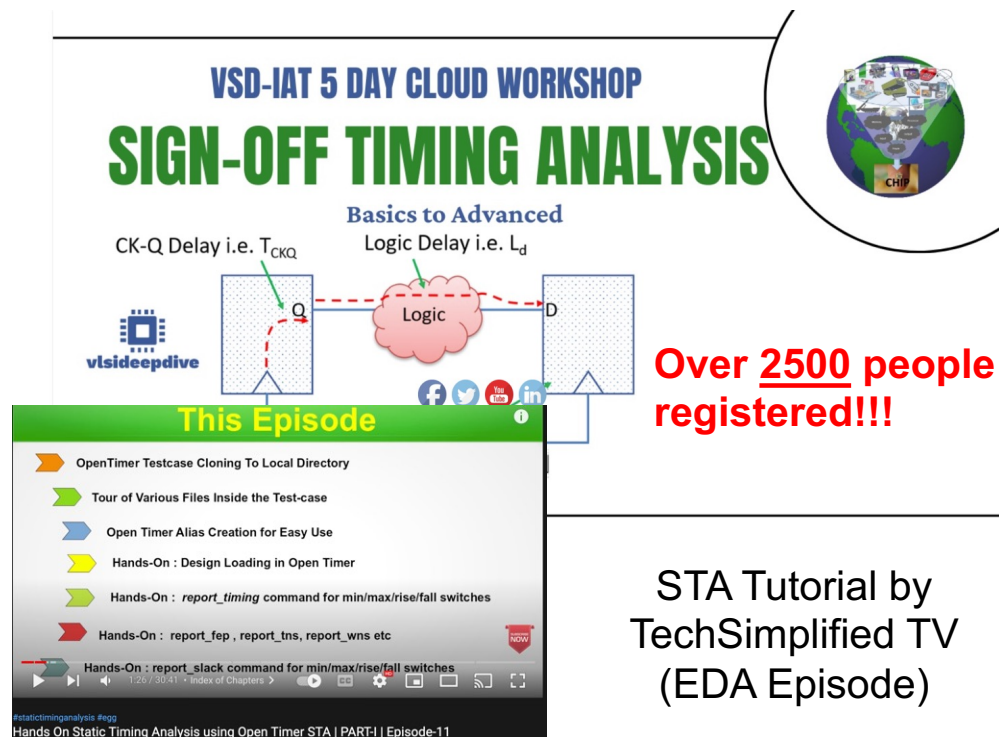
User community



efabless.com



VSD Sign-off Timing Analysis Workshop using OpenTimer: <https://www.vlsisystemdesign.com/sign-off-timing-analysis-basics-to-advanced/>



VSD-IAT 5 DAY CLOUD WORKSHOP  
**SIGN-OFF TIMING ANALYSIS**  
Basics to Advanced

CK-Q Delay i.e.  $T_{CKQ}$  Logic Delay i.e.  $L_d$

Over 2500 people registered!!!

STA Tutorial by TechSimplified TV (EDA Episode)

This Episode

- OpenTimer Testcase Cloning To Local Directory
- Tour of Various Files Inside the Test-case
- Open Timer Alias Creation for Easy Use
- Hands-On : Design Loading in Open Timer
- Hands-On : `report_timing` command for min/max/rise/fall switches
- Hands-On : `report_fep`, `report_tns`, `report_wns` etc
- Hands-On : `report_slack` command for min/max/rise/fall switches

Hands On Static Timing Analysis using Open Timer STA | PART-I | Episode-11