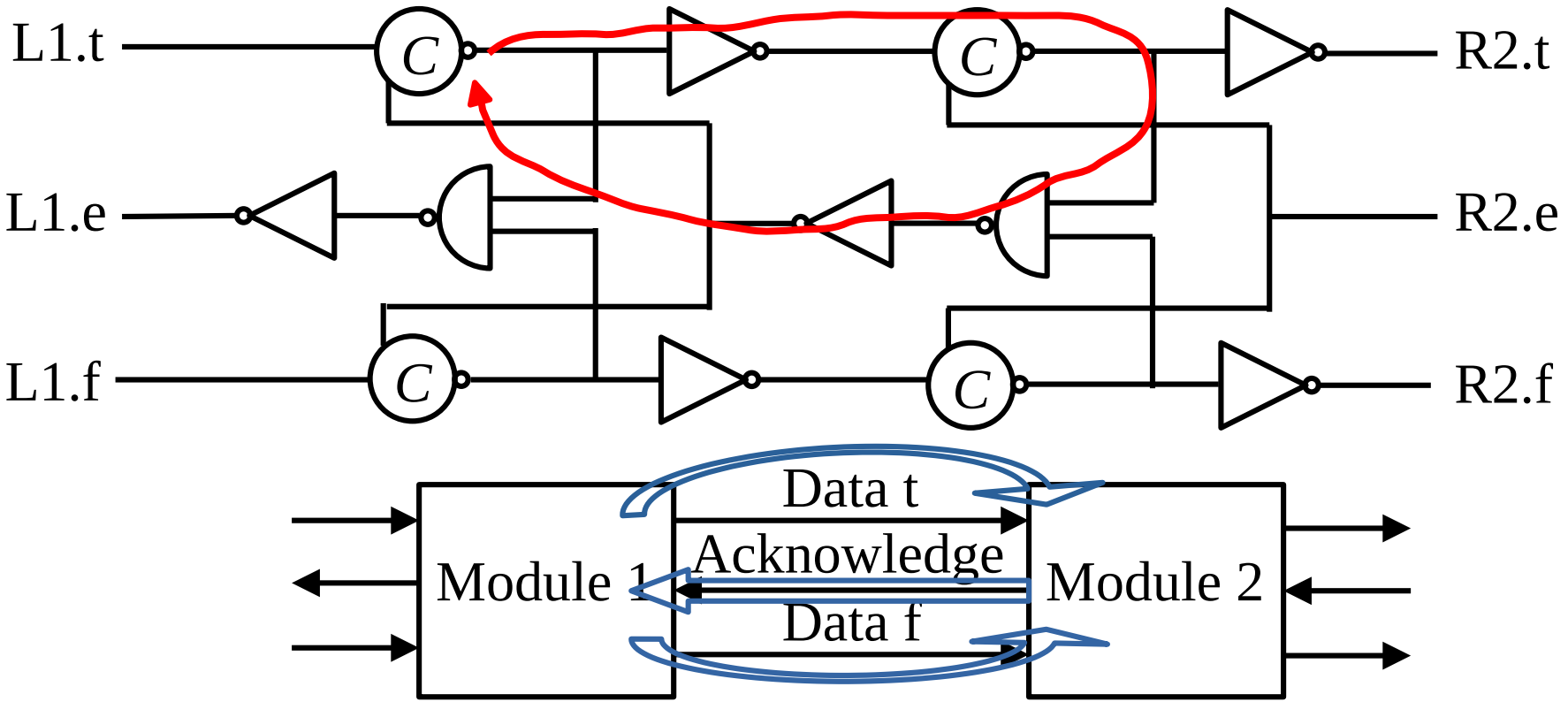
Cyclone: A Static Timing and Power Engine for Asynchronous Circuits

Wenmian Hua[1], Yi-Shan Lu[2], Keshav Pingali[2], Rajit Manohar[1]

[1]Yale University
[2]University of Texas at Austin

Cycles in Asynchronous Circuits



Static Timing Analysis Flow in Cyclone

- Handles cycles explicitly w/o breaking them.
  - Constructs a repetitive event rule (RER) system as the timing graph for an input circuit.
- Solves three key graph problems given a cyclic timing graph.
  - Steady-state delay calculation.
  - Performance as maximum cycle ratio.
  - Event timing propagation.
- Shortens Cyclone runtime by parallelizing key graph algorithms.
  - Parallelism in graph algorithms is data-dependent.
  - Use the operator formulation to capture such parallelism.

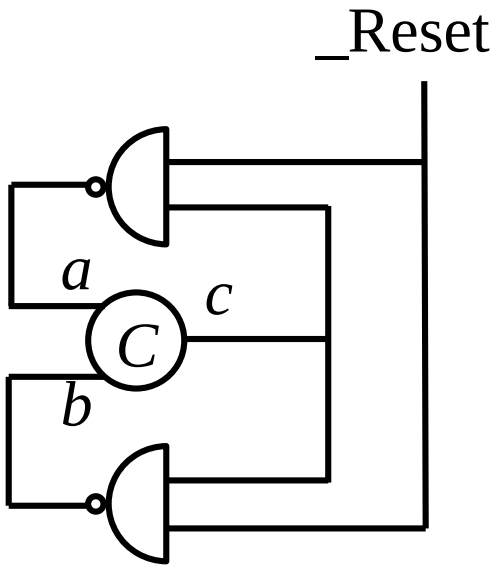
Our Contributions

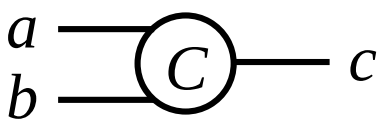
- Construct a full RER system hierarchically for scalability.
- Compute exact critical cycle ratio.
- Define timing quantities for cyclic circuits.
  - Use a critical cycle as performance reference point.
  - Abstract timing constraints as timing forks.
- Parallelize timing algorithms effectively.
  - Steady-state delay computation.
  - Sub-steps in YTO algorithm for max cycle ratio.
  - Event timing propagation.

Outline

- **Timing Graph as RER System**
- Graph Problems to Solve
  - Delay Annotation
  - Performance as Max Cycle Ratio
  - Event Timing
- Parallelizing Graph Algorithms in Cyclone
- Experimental Results
- Conclusions and Future Work

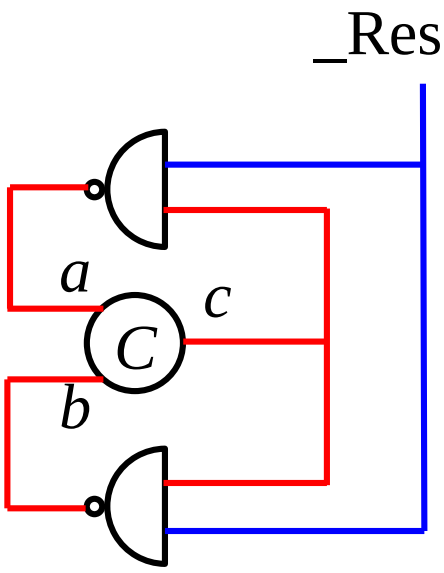
Circuit Example

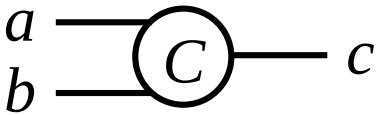




a	*b*	*c*
0	0	0
0	1	c_{prev}
1	0	c_{prev}
1	1	1

Initial State of the Circuit

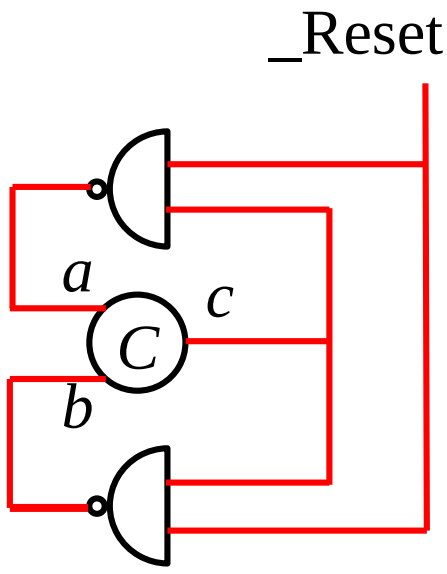




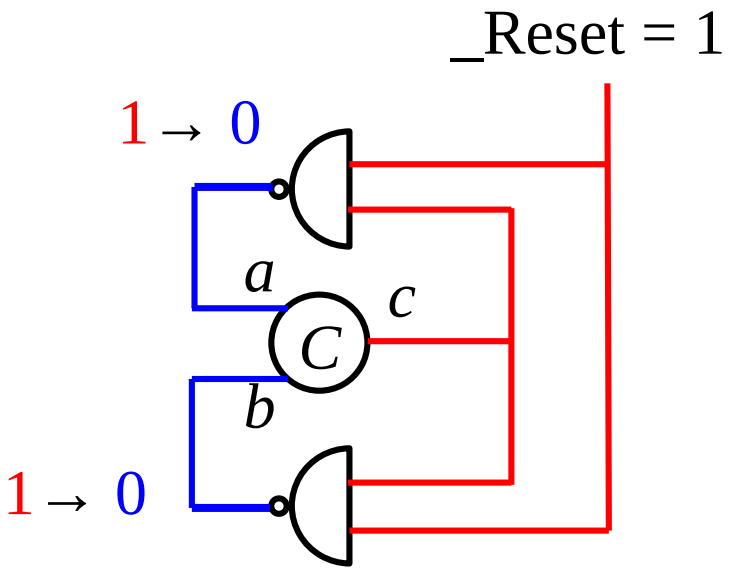
a	b	c
0	0	0
0	1	c_{prev}
1	0	c_{prev}
1	1	1

- _Reset = 0 initially to reset the circuit
  - $a = 1, b = 1, c = 1$

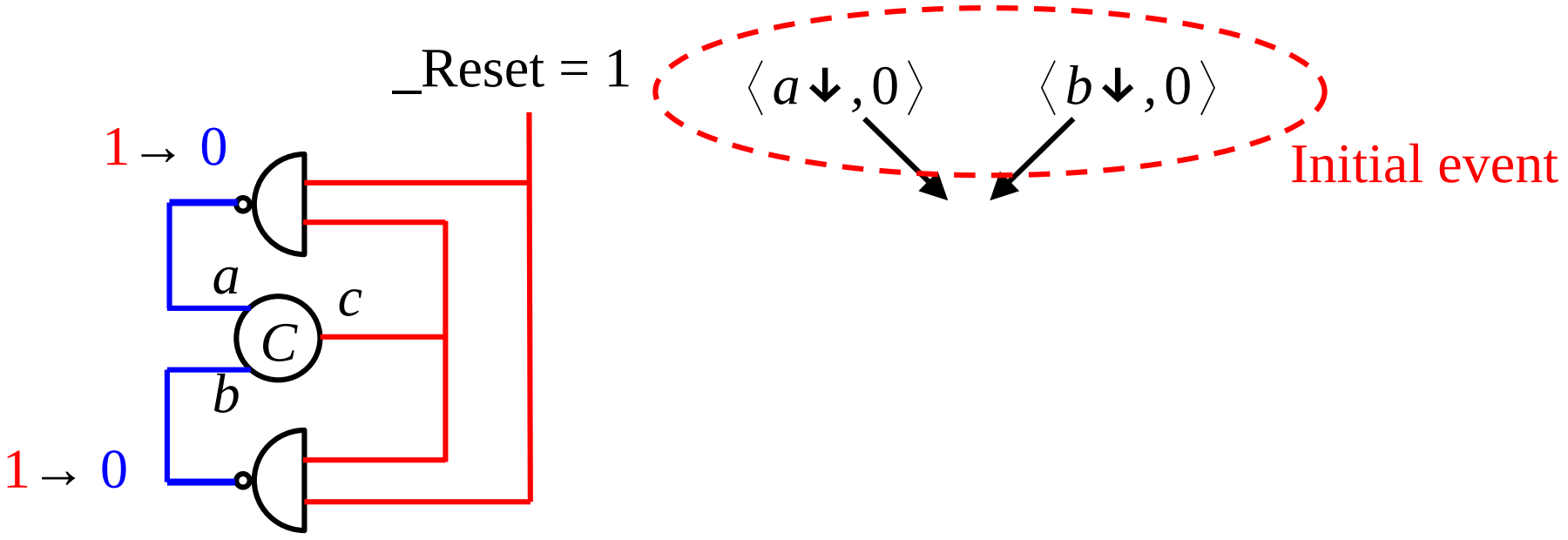
Event Rule (ER) System via Simulation



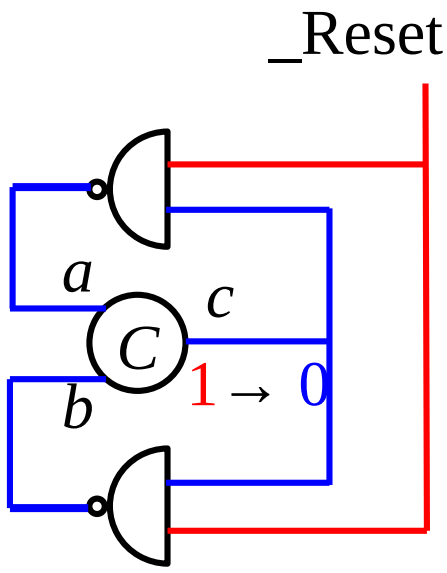
Event Rule (ER) System via Simulation

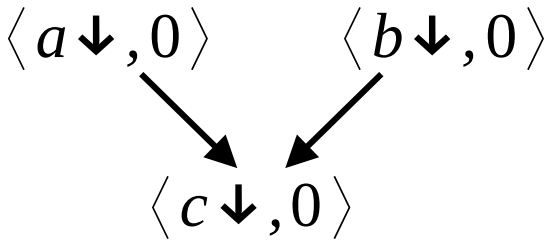


Event Rule (ER) System via Simulation

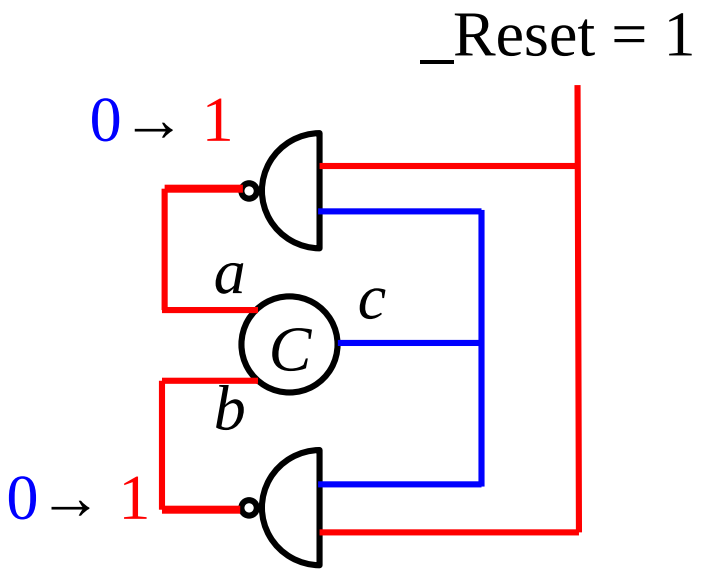


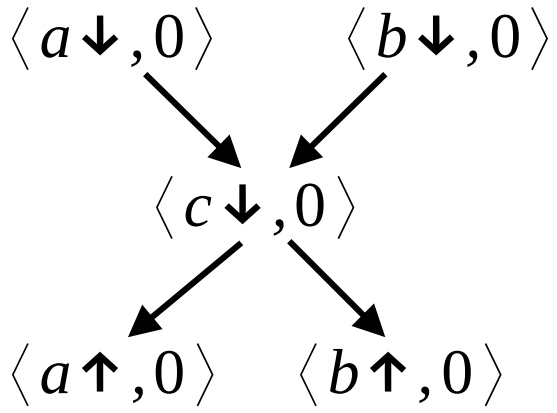
Event Rule (ER) System via Simulation



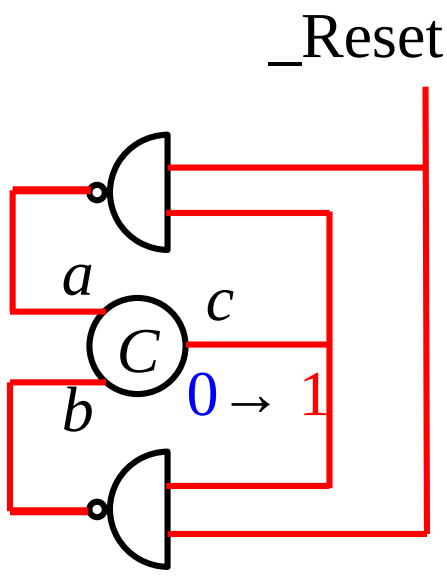


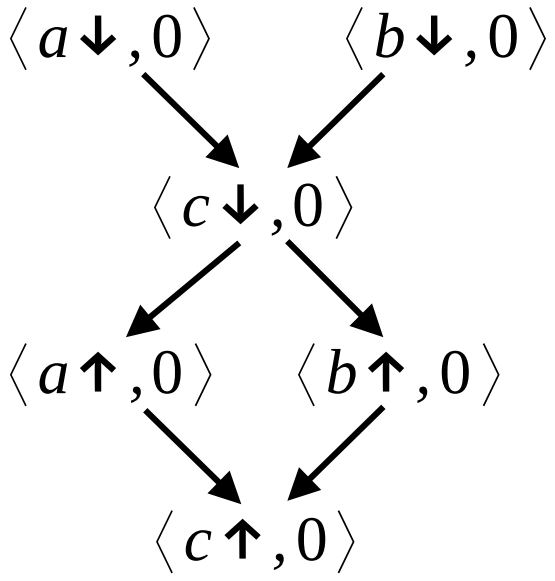
Event Rule (ER) System via Simulation



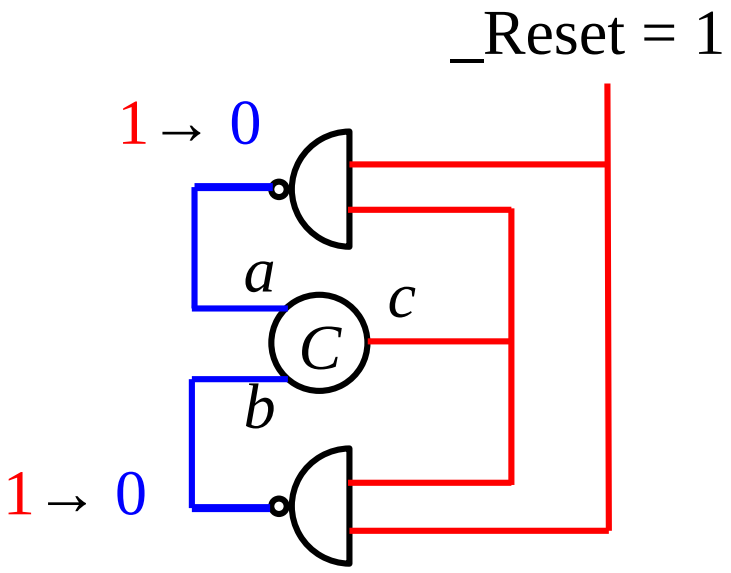


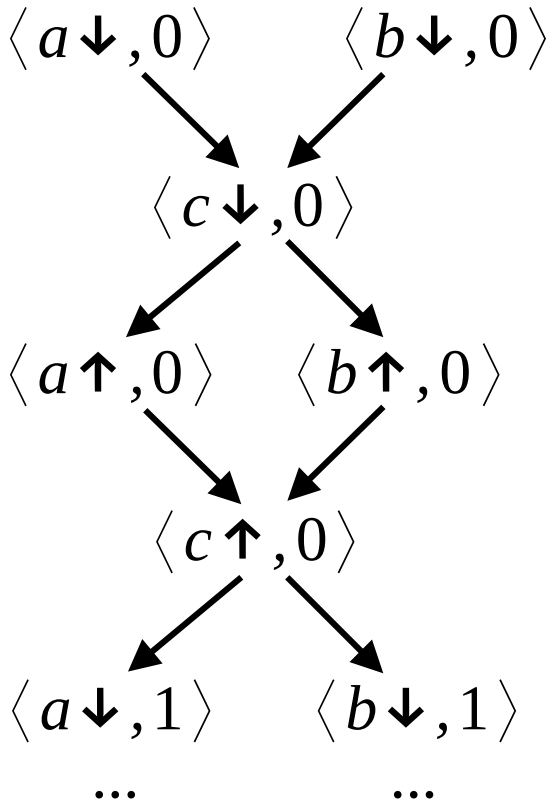
Event Rule (ER) System via Simulation



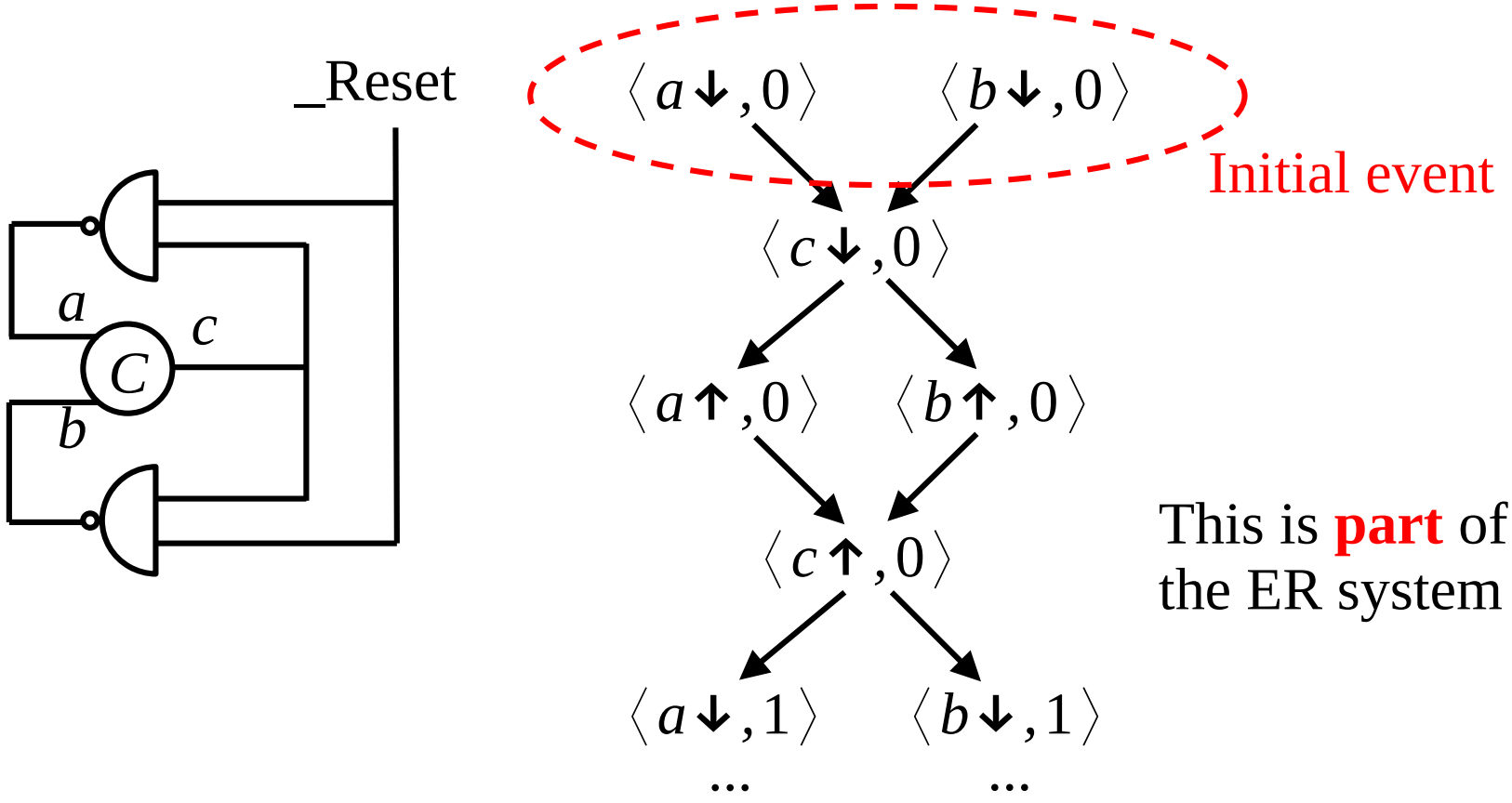


Event Rule (ER) System via Simulation



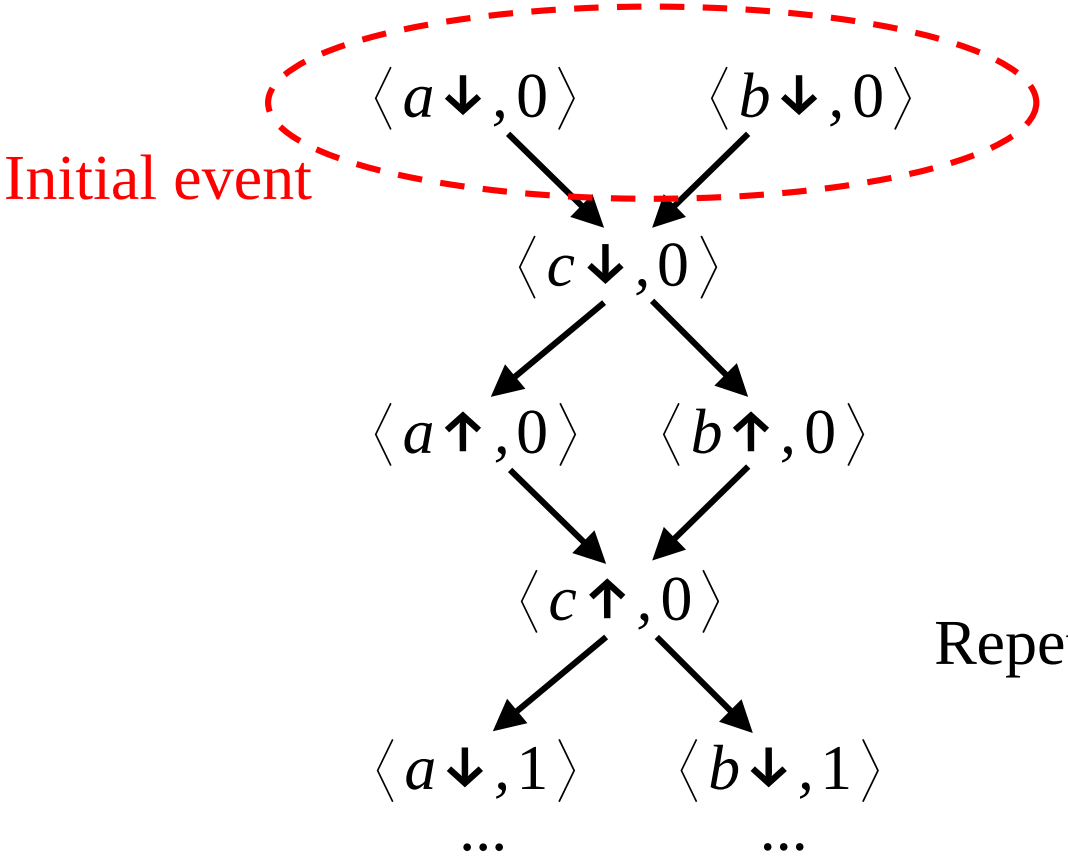


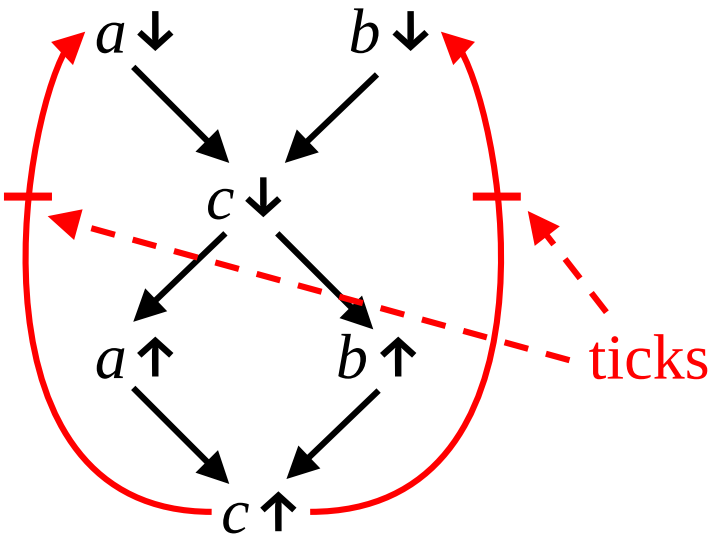
Event Rule (ER) System via Simulation



This is **part** of the ER system

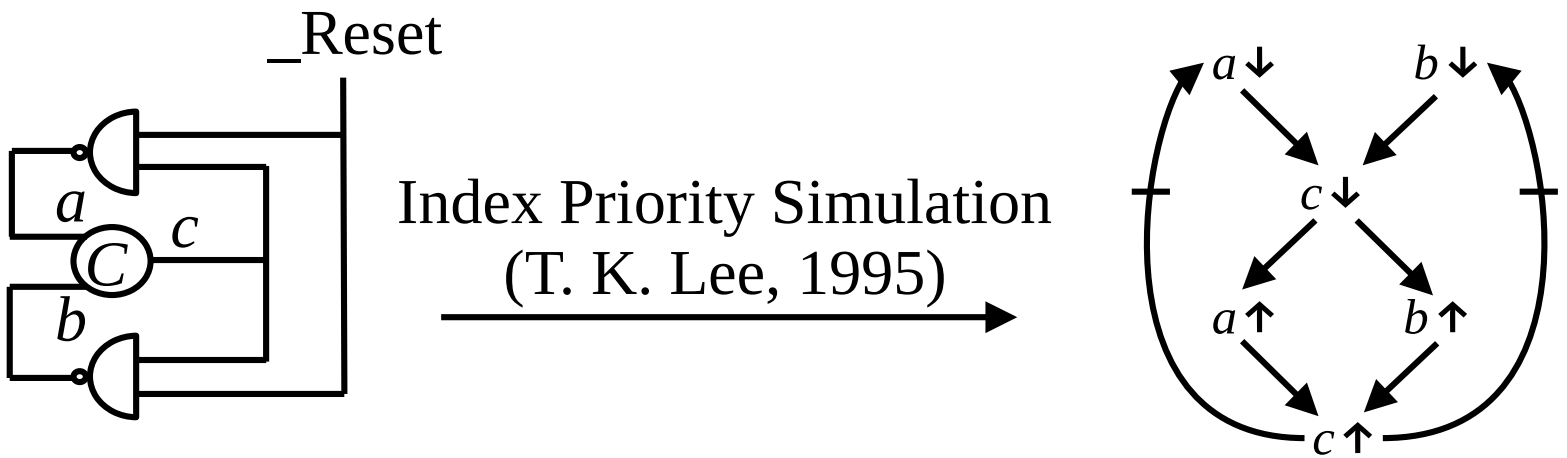
Async Timing Graph – RER System





Repetitive Event Rule (RER) System
(S. M. Burns, 1991)

Hierarchical Construction of RER Systems

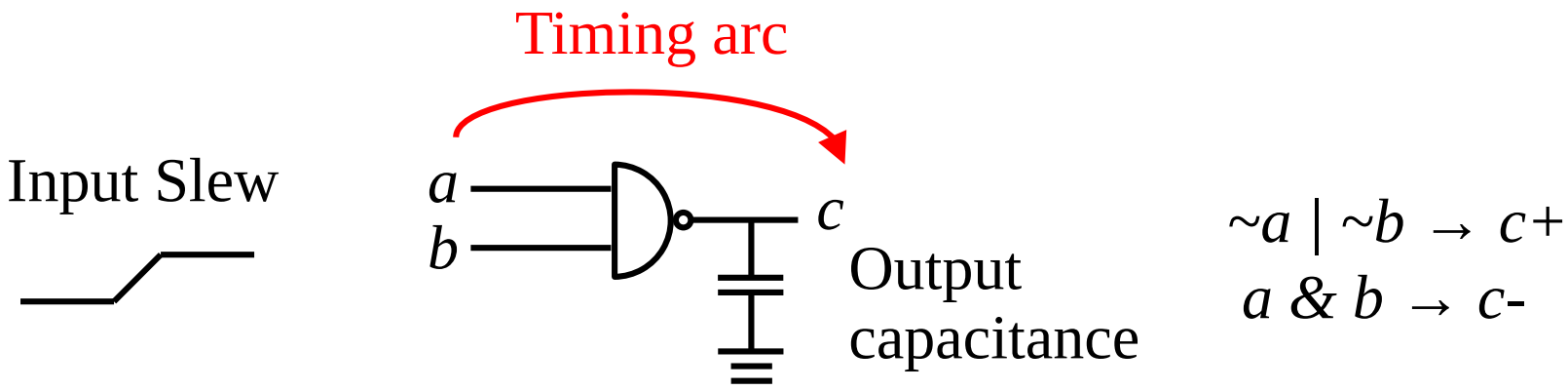


- Full circuit simulation is time consuming.
- Cyclone constructs a full RER system hierarchically.
  - Only simulates circuit processes w/ no sub-processes.
  - Combines the RER systems for sub-processes to get the RER systems for upper-level processes.

Outline

- Timing Graph as RER System
- **Graph Problems to Solve**
  - **Delay Annotation**
  - Performance as Max Cycle Ratio
  - Event Timing
- Parallelizing Graph Algorithms in Cyclone
- Experimental Results
- Conclusions and Future Work

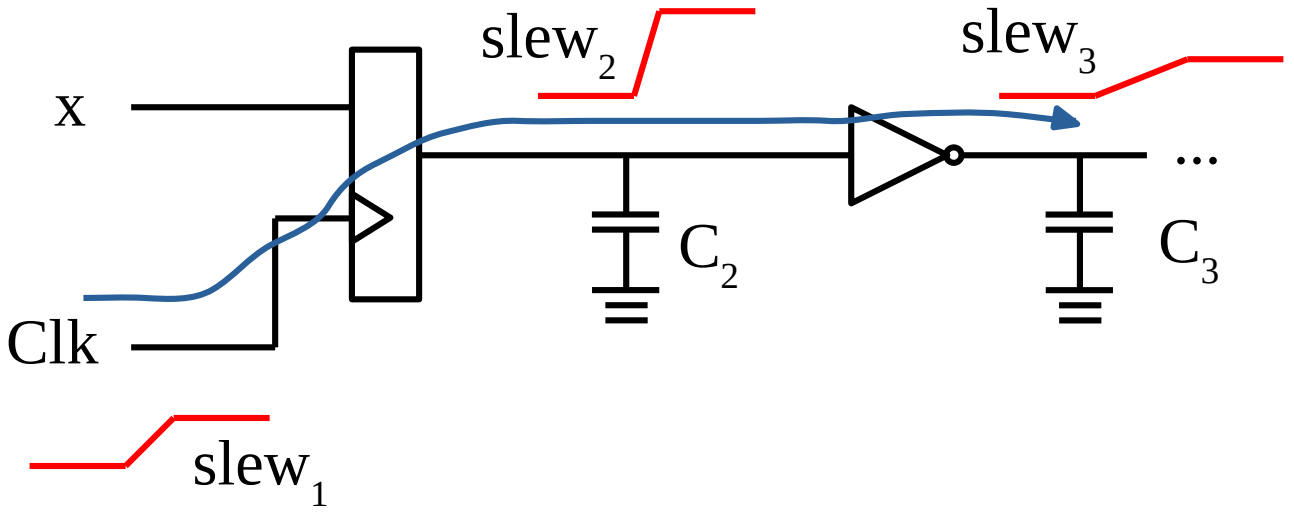
Non-linear Delay/Power Model (NLDM/NLPM)



- Described in Synopsys Liberty files.
- Each timing arc has a propagation delay value, an output slew value, and a dynamic power value.
- Propagation delay, output slew and power depend on input slew and output capacitance.

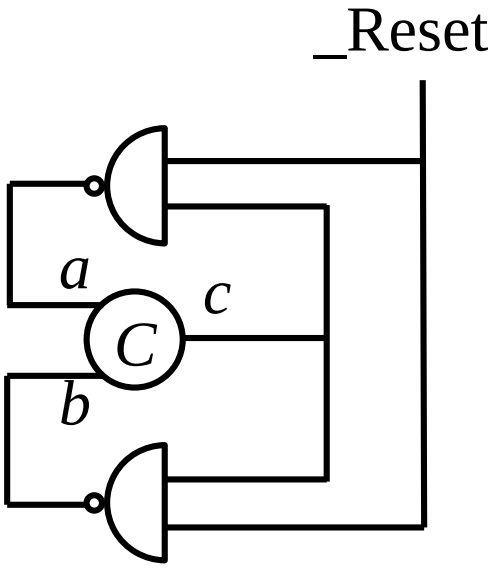
Slew Rate Propagation

- Synchronous circuits are modeled as directed, acyclic graphs.
  - Input slew is known, related to clock parameters.
- All delays/slews can be calculated in sequence starting from the clock pin (following topological order).



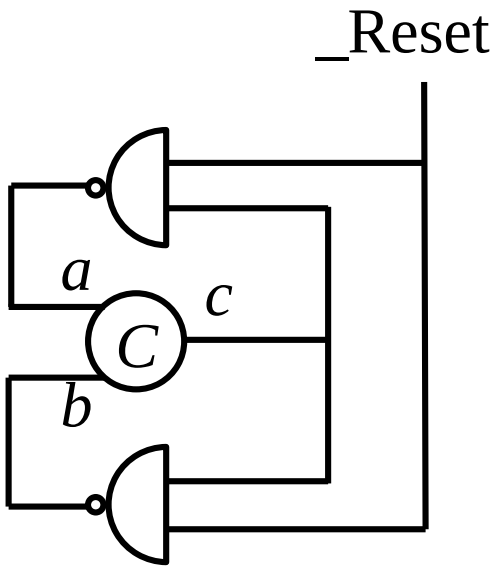
Slew Rate Propagation

- Asynchronous circuits: no clock, cyclic dependence, what to do?



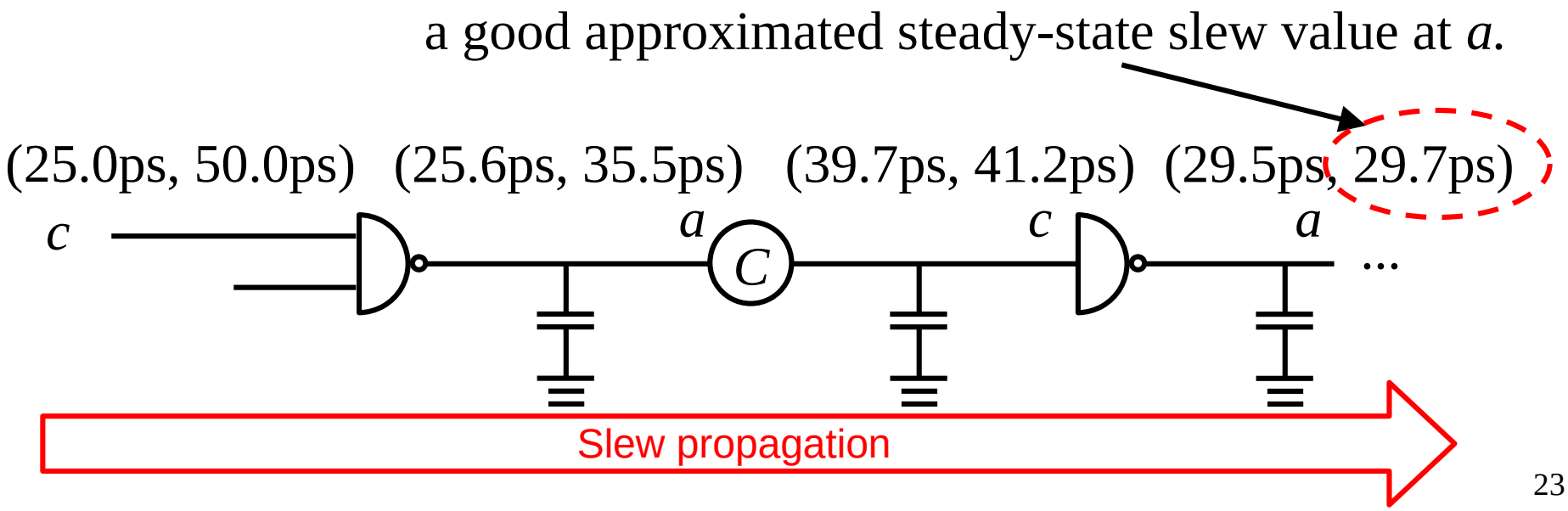
Slew Rate Propagation

- Fact: both delay & output slew are weak functions of input slew.

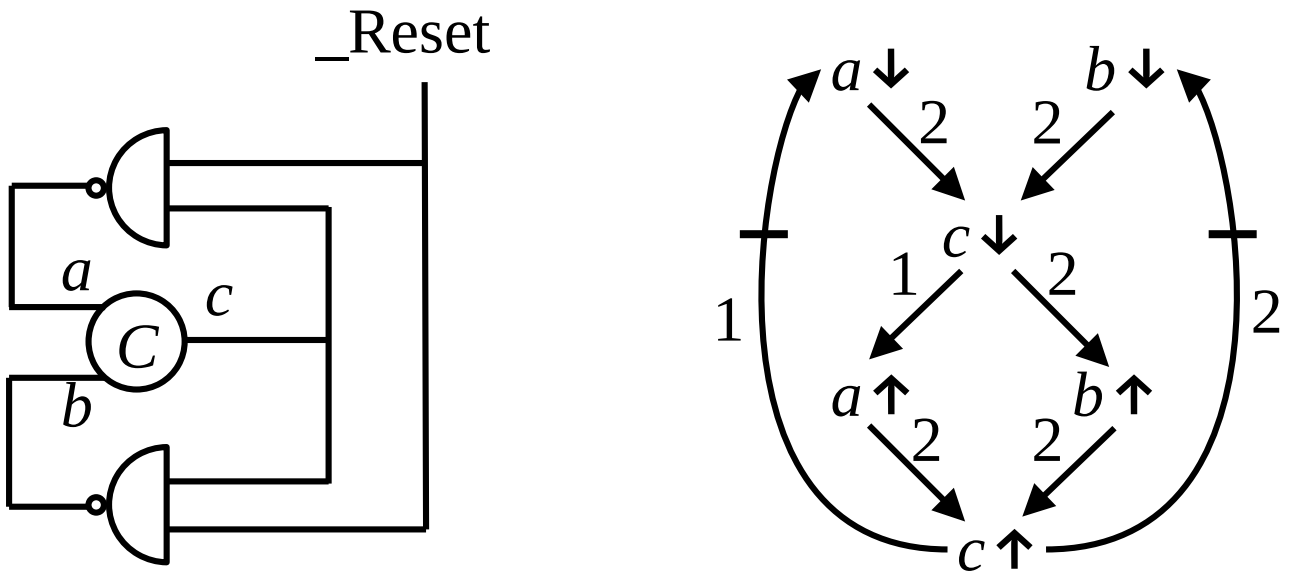


Steady-State Slew Calculation

- Start from an arbitrary point with two extreme slew values, propagate slew intervals through gates.
- The difference will become very small after several stages.



Complete RER Systems

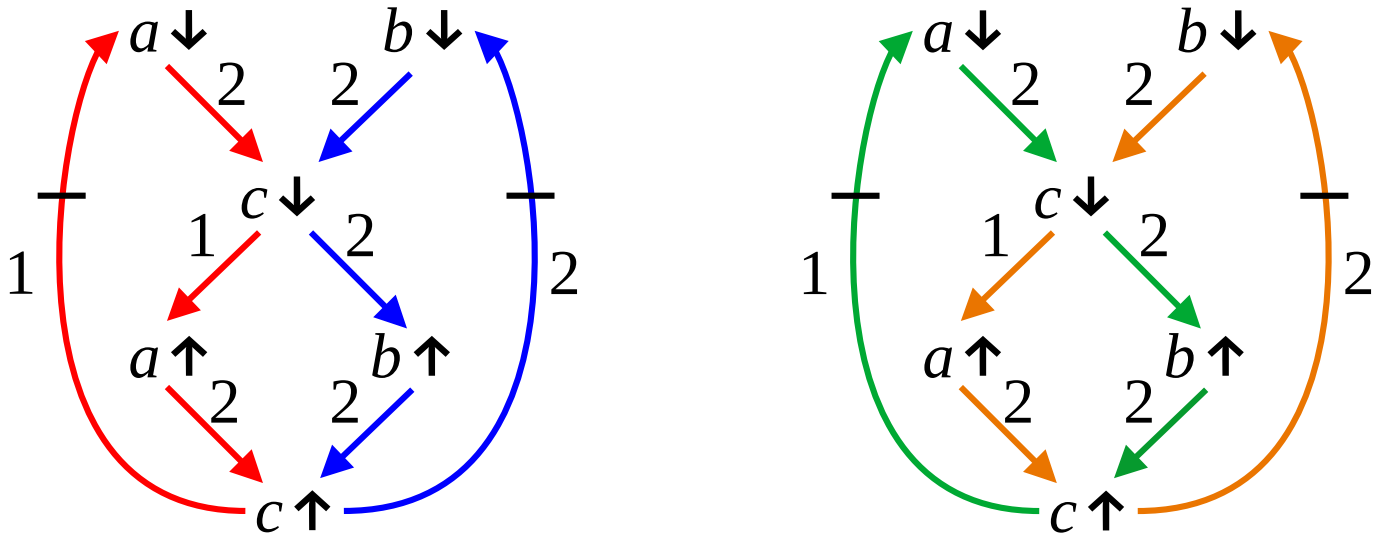


- Edge weights are steady-state delays.
  - Computed using steady-state slew rates.

Outline

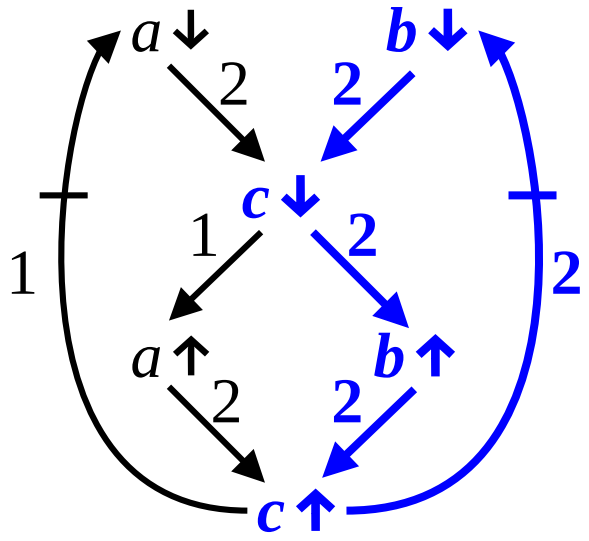
- Timing Graph as RER System
- **Graph Problems to Solve**
  - Delay Annotation
  - **Performance as Max Cycle Ratio**
  - Event Timing
- Parallelizing Graph Algorithms in Cyclone
- Experimental Results
- Conclusions and Future Work

Cycle Ratio of RER Systems



- The cycle ratio of a simple cycle is the total weight divided by the total number of ticks (ε) on all edges of that cycle.

Maximum Cycle Ratio of RER Systems



- A critical cycle is the cycle with the maximum cycle ratio p^* in the graph.
- Validness requirement: every cycle in the graph has at least 1 tick.

Steady-State Exact Periodicity of RER System

- In a valid and strongly connected RER system, there exists integers M and k^* such that for all transitions s and integers $n \geq k^*$,

$$\hat{t}(\langle s, n+M \rangle) - \hat{t}(\langle s, n \rangle) = Mp^*$$

  - M: occurrence period.
  - p^*: maximum cycle ratio, corresponding to circuit cycle period.

- Young-Tarjan-Orlin (YTO) algorithm (using binary heap): $O(|V||E|\log|V|)$
  - Fast for real circuits.

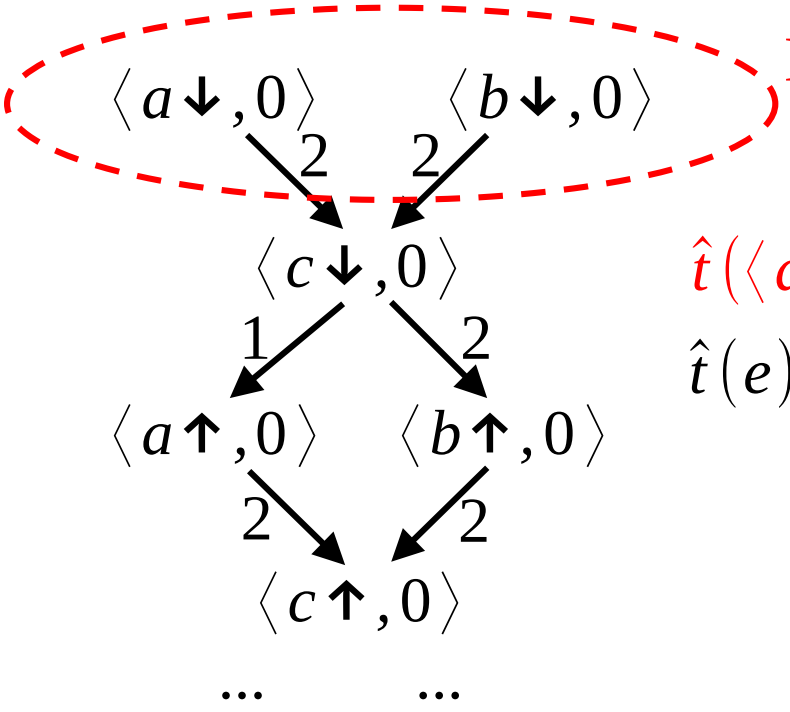
Max Cycle Ratio as Performance Metric

- Critical cycle is the slowest component in the circuit.
- Assume that circuit elements wait for the slowest component.
  - Similar to computing setup-time arrival times in synchronous timing.
- Finally everything is locked to that period.

Outline

- Timing Graph as RER System
- **Graph Problems to Solve**
  - Delay Annotation
  - Performance as Max Cycle Ratio
  - **Event Timing**
- Parallelizing Graph Algorithms in Cyclone
- Experimental Results
- Conclusions and Future Work

Timing of Events in ER Systems

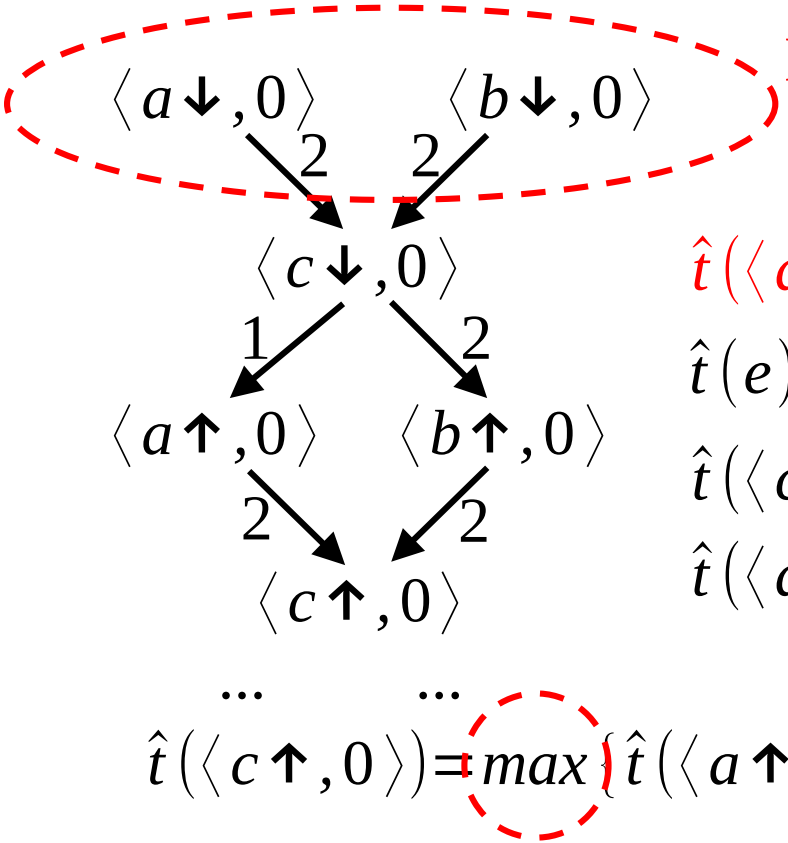


Initial event

$$\hat{t}(\langle a\downarrow,0\rangle)=0,\hat{t}(\langle b\downarrow,0\rangle)=0$$

$$\hat{t}(e)=max\{\hat{t}(e')+d(e',e)\,|\,(e',e)\in E\}$$

Timing of Events in ER Systems



Initial event

$$\hat{t}(\langle a\downarrow,0\rangle)=0,\hat{t}(\langle b\downarrow,0\rangle)=0$$

$$\hat{t}(e)=max\{\hat{t}(e')+d(e',e)\,|\,(e',e)\in E\}$$

$$\hat{t}(\langle c\downarrow,0\rangle)=2,$$

$$\hat{t}(\langle a\uparrow,0\rangle)=3,\hat{t}(\langle b\uparrow,0\rangle)=4,$$

$$\hat{t}(\langle c\uparrow,0\rangle)=max\{\hat{t}(\langle a\uparrow,0\rangle)+2,\hat{t}(\langle b\uparrow,0\rangle)+2\}=6$$

AND-causality

Arrival Time t_{arr}

- We only consider steady-state.
- $t_{arr} = 0$ for initial events on a critical cycle.
- For all other events,
  $$t_{arr}(\langle s, n \rangle) = max\{ t_{arr}(\langle s', n' \rangle) + d(s', s) \mid (s', s) \in E \}$$
  - $n' = n$ if edge (s', s) is not ticked.
  - $n' = n - 1$ if edge (s', s) is ticked.
- Power consumption can be calculated simultaneously.
  - Max of dynamic power on incoming edges.

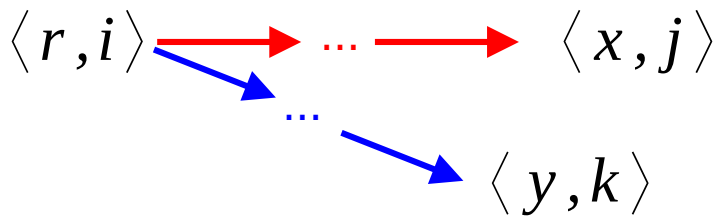
Performance Slack t_s

- Computing t_{req}
  - Events on a critical cycle: $t_{req} = t_{arr}$
    - Critical cycle is the slowest component in the circuit.
  - Other events: $t_{req}(\langle s, n \rangle) = min\{ t_{req}(\langle s', n' \rangle) - d(s, s') \mid (s, s') \in E \}$
    - $n' = n$ if edge (s, s') is not ticked.
    - $n' = n + 1$ if edge (s, s') is ticked.

- $t_s = t_{req} - t_{arr}$ is the maximum time amount an event can be delayed without performance penalty.
  - Similar to setup-time slack in synchronous timing.

Timing Constraints as Timing Forks

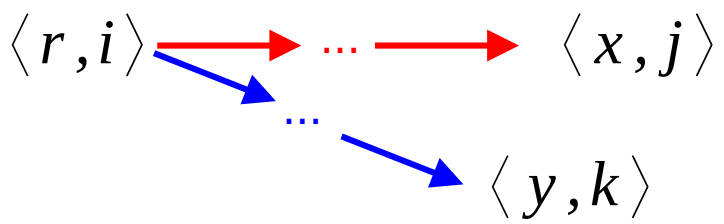
- Cyclone supports the notion of timing forks.
  - To express timing constraints for correct operations of certain circuit families.
- A timing fork $\langle r, i \rangle : \langle x, j \rangle < [margin] \langle y, k \rangle$
  - A root node $\langle r, i \rangle$
  - A slow path $p_{\langle r,i \rangle, \langle x,j \rangle}$ from $\langle r, i \rangle$ to $\langle x, j \rangle$, $j \in \{i, i+1\}$
  - A fast path $p_{\langle r,i \rangle, \langle y,k \rangle}$ from $\langle r, i \rangle$ to $\langle y, k \rangle$, $k \in \{i, i+1\}$



Correctness Slack for Timing Forks

- Timing forks require that the correctness slack ***sl*** satisfies:

$$sl = min\{d(p_{\langle r,i\rangle,\langle x,j\rangle})\} - max\{d(p_{\langle r,i\rangle,\langle y,k\rangle})\} \geq margin$$



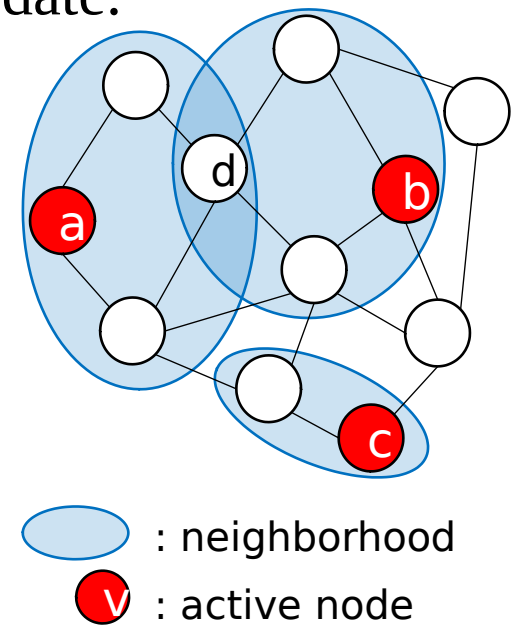
- Different asynchronous logic families have different timing forks.
  - Generated during logic synthesis.
- Similar to hold-time slack in synchronous timing.

Outline

- Timing Graph as RER System
- Graph Problems to Solve
  - Delay Annotation
  - Performance as Max Cycle Ratio
  - Event Timing
- **Parallelizing Graph Algorithms in Cyclone**
- Experimental Results
- Conclusions and Future Work

Operator Formulation

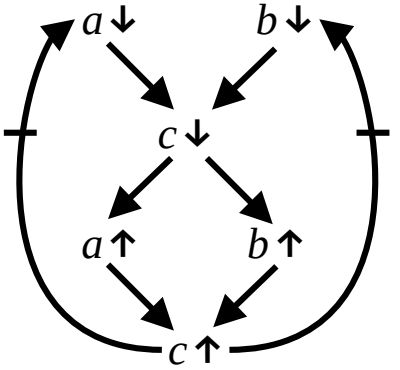
- Active elements: subgraphs where computation is needed.
- Operator: Computation at active elements.
  - Neighborhood: set of nodes/edges touched by the update.
- Schedules: Order to apply operators on active elements.
  - May be constrained for correctness.
  - Some ordering may perform better than the others.
- Activation patterns of algorithms.
  - Topology-driven: all nodes/edges are active.
  - Data-driven: track active nodes/edges w/ worklists.
- Parallelism in disjoint updates.



Shared-memory Galois

- A C++ library for operator formulation.
- Features:
  - Parallel data structures: graphs, bags, etc.
  - Parallel loops over active elements: for_each, do_all, etc.
  - Support for load balancing, scheduling, dynamic work, transactional execution.
- Successes in EDA: FPGA routing [Moctar & Brisk, DAC 2014], AIG rewriting [Possani et al., ICCAD 2018], ASIC global routing [He et al., ICCAD 2019].

Parallelizing Steady-State Slew Calculation



- In non-linear delay model (NLDM),
  - Slew = *f*(predecessors' slew & successors' load).
- Steady-state slew: convergence by Jacobi iterations.
  - Initially, every pin assumes slew range [0, ρ].
  - Each pin computes its new slew range based on its predecessors' old ones.
  - Repeat until the slew ranges are small enough, e.g., ≤ ρ/1,000.
- Topology-driven, unordered algorithms.
  - Every node is active in one round.
  - Same answer by any order of node computation in a round.

Parallelizing YTO Algorithm (1)

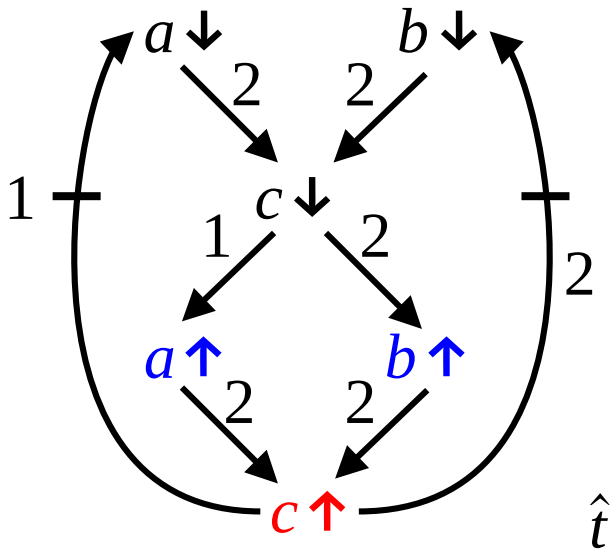
- Longest-path tree construction:
  - Ignore all ticked edges.
  - Construct a longest-path tree from a synthetic source to all nodes.
- Data-driven, unordered algorithm.
  - Use a worklist to track active nodes whose distances change.
  - Max and addition are commutative
    => same answer for any order of distance updates.

Parallelizing YTO Algorithm (2)

- Edge swapping:
  - Pick the edge w/ largest path ratio (ordered by a max heap).
  - Check if a cycle is formed in the longest-path tree.
    - If so, report the path ratio as max cycle ratio.
    - Otherwise, repeat after updating longest-path tree & max heap.

- Data-driven, ordered algorithm.
  - Track active edges with a max heap.
  - Computation is done only for the root of the max heap at a time.

Parallelizing Arrival Time Computation



$$\hat{t}(\langle a\uparrow, 0\rangle) = 3 \qquad \hat{t}(\langle b\uparrow, 0\rangle) = 4$$

$$\hat{t}(\langle c\uparrow, 0\rangle) = max\{\hat{t}(\langle a\uparrow, 0\rangle) + 2, \hat{t}(\langle b\uparrow, 0\rangle) + 2\} = 6$$

Data-driven, unordered algorithm

Outline

- Timing Graph as RER System
- Graph Problems to Solve
  - Delay Annotation
  - Performance as Max Cycle Ratio
  - Event Timing
- Parallelizing Graph Algorithms in Cyclone
- **Experimental Results**
- Conclusions and Future Work

Experimental Setup

- Platform
  - Compiler: g++ 8.1
  - Libraries: boost 1.67 and Galois 5.0
  - CentOS 7
  - 4 14-core Intel Xeon Gold 5120 2.2 GHz CPUs (56 cores in total)
  - 187 GB memory
- Comparison: against linear programming by CPLEX

Benchmark Circuits

Circuit	Family	#Pins	p^* (ns)	M	Power (mW)
bd203	Bundled-data	495	0.44	1	0.44
c2670	QDI	22,171	1.65	5	7.54
ac97ctrl	QDI	650,709	3.79	3	99.70
s38584	QDI	807,903	9.68	1	48.81
aescore	QDI	1,017,817	8.61	1	69.49
vgalcd	QDI	5,689,435	7.05	1	473.20

Runtime for Finding Max Cycle Ratio

All runtimes are in seconds.

Circuit	#Pins	LP time	YTO time	$\#thread_{YTO}$	speedup
bd203	495	<0.01	0.01	1	--
c2670	22,171	0.69	0.44	1	1.57
ac97ctrl	650,709	79.93	8.87	21	9.01
s38584	807,903	79.39	3.92	56	20.25
aescore	1,017,817	74.44	4.76	49	15.64
vgalcd	5,689,435	2,959.01	105.48	56	28.05

Cyclone End-to-end Runtime

Includes steady-state delay calculation, YTO, and timing propagation.
All runtimes are in seconds.

Circuit	#Pins	sequential time	best time	$\#thread_{best}$	speedup
bd203	495	0.03	0.03	1	1.00
c2670	22,171	1.69	0.86	7	1.97
ac97ctrl	650,709	102.53	15.54	35	6.60
s38584	807,903	51.07	9.58	35	5.33
aescore	1,017,817	95.30	12.62	42	7.55
vgalcd	5,689,435	2,994.24	159.58	56	18.76

Outline

- Timing Graph as RER System
- Graph Problems to Solve
  - Delay Annotation
  - Performance as Max Cycle Ratio
  - Event Timing
- Parallelizing Graph Algorithms in Cyclone
- Experimental Results
- **Conclusions and Future Work**

Conclusions

- We present Cyclone, the first practical asynchronous static timing and power analysis engine.
  - Hierarchical construction of full RER system.
  - Steady-state slew rate computation.
  - Exact max cycle ratio computation.
  - Definition of arrival time, required time, performance slack and correctness slack for cyclic circuits.
- We effectively parallelize graph algorithms in Cyclone:
  - Steady-state slew propagation.
  - Sub-steps in YTO algorithm.
  - Timing propagation.

Future Work

- General timing graph construction.
  - Support for more circuit families.
  - Support for OR-causality.
- More advanced delay model – composite current source (CCS).
- Hierarchical timing analysis.
- Top-k critical cycle report.

Thanks!

Questions?