Cyclone: A Static Timing and Power Engine for Asynchronous Circuits

Wenmian Hua\textsuperscript{1}, Yi-Shan Lu\textsuperscript{2}, Keshav Pingali\textsuperscript{2}, Rajit Manohar\textsuperscript{1}

\textsuperscript{1}Yale University
\textsuperscript{2}University of Texas at Austin
Cycles in Asynchronous Circuits

L1.t

L1.e

L1.f

R2.t

R2.e

R2.f

Module 1

Module 2

Data t

Acknowledge

Data f
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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$c_{\text{prev}}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$c_{\text{prev}}$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Initial State of the Circuit

- \_Reset = 0 initially to reset the circuit
- \( a = 1, \ b = 1, \ c = 1 \)
Event Rule (ER) System via Simulation

_Reset = 1
Event Rule (ER) System via Simulation

Reset = 1

1 → 0

1 → 0
Event Rule (ER) System via Simulation

\[ \langle a \downarrow, 0 \rangle \quad \langle b \downarrow, 0 \rangle \]

Initial event

_C Reset = 1

1 → 0

1 → 0
Event Rule (ER) System via Simulation

_C reset = 1

\( \langle a \downarrow, 0 \rangle, \langle b \downarrow, 0 \rangle, \langle c \downarrow, 0 \rangle \)
Event Rule (ER) System via Simulation

Reset = 1

\[ \langle a \downarrow, 0 \rangle \quad \langle b \downarrow, 0 \rangle \]

\[ \langle c \downarrow, 0 \rangle \]

\[ \langle a \uparrow, 0 \rangle \quad \langle b \uparrow, 0 \rangle \]
Event Rule (ER) System via Simulation

\_Reset = 1

\[ \langle a \downarrow, 0 \rangle \quad \langle b \downarrow, 0 \rangle \]

\[ \langle c \downarrow, 0 \rangle \quad \langle a \uparrow, 0 \rangle \quad \langle b \uparrow, 0 \rangle \quad \langle c \uparrow, 0 \rangle \]
Event Rule (ER) System via Simulation

\_Reset = 1

\begin{align*}
\langle a \downarrow, 0 \rangle & \rightarrow \langle b \downarrow, 0 \rangle \\
\langle c \downarrow, 0 \rangle & \\
\langle a \uparrow, 0 \rangle & \rightarrow \langle b \uparrow, 0 \rangle \\
\langle c \uparrow, 0 \rangle & \\
\langle a \downarrow, 1 \rangle & \rightarrow \langle b \downarrow, 1 \rangle \\
\ldots & \rightarrow \ldots
\end{align*}
Event Rule (ER) System via Simulation

Initial event

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.

\[(25.0\text{ps}, 50.0\text{ps}) \quad (25.6\text{ps}, 35.5\text{ps}) \quad (39.7\text{ps}, 41.2\text{ps}) \quad (29.5\text{ps}, 29.7\text{ps})\]

a good approximated steady-state slew value at \(a\).
• 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
The cycle ratio of a simple cycle is the total weight divided by the total number of ticks ($\varepsilon$) 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

\[
\begin{align*}
\langle a \downarrow, 0 \rangle & \quad \langle b \downarrow, 0 \rangle \\
\langle c \downarrow, 0 \rangle & \quad \langle a \uparrow, 0 \rangle & \quad \langle b \uparrow, 0 \rangle \\
\langle c \uparrow, 0 \rangle
\end{align*}
\]

Initial event

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

\[
\hat{t}(e) = \max \{ \hat{t}(e') + d(e', e) \mid (e', e) \in E \}
\]
Timing of Events in ER Systems

\[
\begin{align*}
\langle a \downarrow, 0 \rangle & \quad \langle b \downarrow, 0 \rangle \\
\langle c \downarrow, 0 \rangle & \quad 2 \quad 2 \quad \langle b \uparrow, 0 \rangle \\
\langle a \uparrow, 0 \rangle & \quad \langle b \uparrow, 0 \rangle \\
\langle c \uparrow, 0 \rangle & \quad 2 \quad 2 \\
\ldots & \quad \ldots \\
\hat{t}(\langle c \uparrow, 0 \rangle) & = \max \{ \hat{t}(\langle a \uparrow, 0 \rangle) + 2, \hat{t}(\langle b \uparrow, 0 \rangle) + 2 \} = 6
\end{align*}
\]

Initial event

\[
\begin{align*}
\hat{t}(\langle a \downarrow, 0 \rangle) = 0, \hat{t}(\langle b \downarrow, 0 \rangle) = 0 \\
\hat{t}(e) = \max \{ \hat{t}(e') + d(e', e) \mid (e', e) \in E \} \\
\hat{t}(\langle c \downarrow, 0 \rangle) = 2, \\
\hat{t}(\langle a \uparrow, 0 \rangle) = 3, \hat{t}(\langle b \uparrow, 0 \rangle) = 4,
\end{align*}
\]

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,
  \[ t_{arr}(\langle s, n \rangle) = \max \{ t_{arr}(\langle s', n' \rangle) + d(s', s) | (s', s) \in E \} \]
  - $n' = n$ 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 \( \langle r, i \rangle : \langle x, j \rangle < \text{[margin]} \langle y, k \rangle \)
  - 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 \( \langle y, k \rangle \), \( k \in \{i, i+1\} \)

\[
\langle r, i \rangle \quad \rightarrow \quad ... \quad \rightarrow \quad \langle x, j \rangle \\
\quad \rightarrow \quad ... \quad \rightarrow \quad \langle y, k \rangle
\]
Correctness Slack for Timing Forks

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

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

- 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 with the 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

\[ t(\langle a \uparrow, 0 \rangle) = 3 \]

\[ t(\langle b \uparrow, 0 \rangle) = 4 \]

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

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

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Family</th>
<th>#Pins</th>
<th>$p^*$ (ns)</th>
<th>$M$</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bd203</td>
<td>Bundled-data</td>
<td>495</td>
<td>0.44</td>
<td>1</td>
<td>0.44</td>
</tr>
<tr>
<td>c2670</td>
<td>QDI</td>
<td>22,171</td>
<td>1.65</td>
<td>5</td>
<td>7.54</td>
</tr>
<tr>
<td>ac97ctrl</td>
<td>QDI</td>
<td>650,709</td>
<td>3.79</td>
<td>3</td>
<td>99.70</td>
</tr>
<tr>
<td>s38584</td>
<td>QDI</td>
<td>807,903</td>
<td>9.68</td>
<td>1</td>
<td>48.81</td>
</tr>
<tr>
<td>aescore</td>
<td>QDI</td>
<td>1,017,817</td>
<td>8.61</td>
<td>1</td>
<td>69.49</td>
</tr>
<tr>
<td>vgalcd</td>
<td>QDI</td>
<td>5,689,435</td>
<td>7.05</td>
<td>1</td>
<td>473.20</td>
</tr>
</tbody>
</table>
### Runtime for Finding Max Cycle Ratio

All runtimes are in seconds.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Pins</th>
<th>LP time</th>
<th>YTO time</th>
<th>#thread&lt;sub&gt;YTO&lt;/sub&gt;</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bd203</td>
<td>495</td>
<td>&lt;0.01</td>
<td>0.01</td>
<td>1</td>
<td>--</td>
</tr>
<tr>
<td>c2670</td>
<td>22,171</td>
<td>0.69</td>
<td>0.44</td>
<td>1</td>
<td>1.57</td>
</tr>
<tr>
<td>ac97ctrl</td>
<td>650,709</td>
<td>79.93</td>
<td>8.87</td>
<td>21</td>
<td>9.01</td>
</tr>
<tr>
<td>s38584</td>
<td>807,903</td>
<td>79.39</td>
<td>3.92</td>
<td>56</td>
<td>20.25</td>
</tr>
<tr>
<td>aescore</td>
<td>1,017,817</td>
<td>74.44</td>
<td>4.76</td>
<td>49</td>
<td>15.64</td>
</tr>
<tr>
<td>vgalcd</td>
<td>5,689,435</td>
<td>2,959.01</td>
<td>105.48</td>
<td>56</td>
<td>28.05</td>
</tr>
</tbody>
</table>

All runtimes are in seconds.
Cyclone End-to-end Runtime

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

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