Using RICE for FPGA Routing Delay Annotation

Andrew Clinton, Matt Liberty, Ian Kuon
FPGA Routing (Interconnect)

FPGA routing consists of a network of wires and programmable switches

- Wire is modeled with a reduced RC network
- Drivers are modeled as a SPICE netlist
- 2-Level pass gate mux is modeled with a capacitive load model
- Programmability comes through SRAM bits that control the pass gate switches
Routing Delay Annotation

Routing (interconnect) delay calculation contributes significantly to overall FPGA compiler runtime

- Timing graph topology and wire loading are not known in advance
- Due to this high degree of runtime configurability, we’ve previously relied on high-accuracy SPICE-like simulations to calculate routing delays
- These simulations have historically contributed as much as 10% to overall FPGA compiler runtime
  - For just signoff timing, the proportion of runtime is larger
In software, routing is represented as a forest of trees

- Trees are sourced and sinked at timing cells (such as logic elements or DSPs)
- For each tree, delay annotation traverses the tree in depth-first order
- Each driver/load pair is simulated using SPICE
  - Output waveform(s) are propagated to children
- Node delays are saved
RICE – Rapid Interconnect Evaluation

An implementation of AWE (Asymptotic Waveform Evaluation)

- Black box that takes a circuit as input and provides the impulse response as output
  - In our case, always a grounded RC circuit
  - Sometimes containing resistor loops
- Impulse response is a sum of exponentials
  - Given impulse response, can calculate the output voltage waveform for an arbitrary input
- Generally $O(n)$ in circuit complexity and number of moments
# RICE vs SPICE, 84 Node RC Network Step Response

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Delay (ps)</th>
<th>Error</th>
<th>Runtime (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RICE, order 1</td>
<td>94.554</td>
<td>6%</td>
<td>41.0</td>
</tr>
<tr>
<td>RICE, order 2</td>
<td>108.399</td>
<td>8%</td>
<td>50.9</td>
</tr>
<tr>
<td><strong>RICE, order 3</strong></td>
<td><strong>100.137</strong></td>
<td><strong>&lt;0.01%</strong></td>
<td><strong>57.0</strong></td>
</tr>
<tr>
<td>RICE, order 4</td>
<td>100.139</td>
<td>&lt;0.01%</td>
<td>63.0</td>
</tr>
<tr>
<td>SPICE, 50ps step</td>
<td>99.377</td>
<td>0.75%</td>
<td>143.3</td>
</tr>
<tr>
<td>SPICE, 10ps step</td>
<td>100.180</td>
<td>0.04%</td>
<td>264.6</td>
</tr>
<tr>
<td><strong>SPICE, 5ps step</strong></td>
<td><strong>100.128</strong></td>
<td><strong>&lt;0.01%</strong></td>
<td><strong>418.6</strong></td>
</tr>
<tr>
<td>SPICE, 2ps step</td>
<td>100.135</td>
<td>&lt;0.01%</td>
<td>872.0</td>
</tr>
</tbody>
</table>
Integrating RICE with Non-Linear Drivers

RICE can calculate accurate linear circuit delays approximately 1 order of magnitude faster than our SPICE simulator. However, it doesn’t handle non-linear drivers.

- The challenge is then to obtain sufficiently accurate driver delays without incurring the cost of simulations.

- Our general approach involves pre-computing a table of voltage waveforms at the driver output, parameterized by:
  - Input waveform slew
  - Output load (pi model)

- Similar to Liberty cell models, we will query this table at runtime.
Cumulative Approximation Sequence

The following slides will outline a sequence of approximations that help to break down the sources of error that arise from replacing SPICE with RICE:

- 3.1 Splitting Driver / Load Simulations
- 3.2 Reducing Input Waveforms to 1 Parameter
- 3.3 Using RICE for Loads
- 3.4 Reducing Driver Load Model to 3 Parameters
- 3.5 4D Driver Waveform Cache
- 3.6 2D Driver Waveform Cache
3.1 Splitting Driver and Load Simulations

Driver and load delay calculation need to be separate to substitute RICE for just the load

- As a first step toward this goal, split up the monolithic driver/load simulation into separate driver and load sims
- With a small step size, there should be little impact on delays
- Useful for sanity checking our flow
3.2 Reducing Input Waveforms to 1 Parameter

To key our waveform cache on input waveforms, we need to reduce waveform dimensionality

- Routing Waveforms are strongly exponential
  - We’ve chosen this shape as our fit target
- Some outliers don’t fit well, resulting in bias/variance
Our initial evaluation showed almost no error (<0.01%) for step response

- Calculating the response to arbitrary input waveforms leads to some error due to our convolution implementation
  - We found it necessary to implement this convolution with discretization and an internal 5ps step size to improve runtime

- Low order could compromise accuracy
  - Order 4 seems to converge fairly completely in our tests
3.4 Reducing Driver Load Model to 3 Parameters

To key our waveform cache on the output load, we need to reduce the dimensionality of the load

- A Pi model for the load is readily available from the first 4 moments in RICE
- Some inaccuracy in driver waveform shape is possible with this approximation
Given an input waveform / load in reduced parameter space, we can tabulate driver waveforms:

- Choose evaluation points on each axis
- Evaluate and store *monotonic* waveforms
- At runtime, interpolation/extrapolate waveforms in the cache
  - Interpolating time, not voltage requires monotonicity
3.5 4D Interpolation

Several sources of error creep in with interpolation:

- Interpolation error
  - Choice of evaluation points and cache resolution have a strong influence on error

- Extrapolation error

- Forced monotonicity

- Waveform simplification
  - For efficiency, choose fixed evaluation voltages and use vector CPU instructions
Results

We integrated IRICE (Intel’s implementation of RICE) into our FPGA signoff timing engine in Quartus

- To generate test routes, we compiled a single large user design for the Stratix 10 device, resulting in routing with \( n \approx 1.3 \text{ million} \) routing elements

- Each successive approximation (3.1 – 3.6) was statistically compared to the ground truth for both rising and falling delays
  - Ground truth delays were calculated using our custom SPICE simulator with a small step size (5ps)
  - We also compared against SPICE in the lower accuracy mode (50ps) that we have used in production in the past
Accuracy – Rising Delays

Percent Error

<table>
<thead>
<tr>
<th>3.1 Split Simulations</th>
<th>3.2 Simplify Input Waveforms</th>
<th>3.3 Simulate Load with RICE</th>
<th>3.4 Pi Model for Driver Load</th>
<th>3.5 4D Waveform Cache</th>
<th>3.5 4D Waveform Cache (2x resolution)</th>
<th>3.6 2D Waveform Cache</th>
<th>SPICE, 50ps Maximum Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bias</td>
<td>0.0%</td>
<td>0.5%</td>
<td>0.7%</td>
<td>0.7%</td>
<td>0.9%</td>
<td>0.9%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.1%</td>
<td>0.6%</td>
<td>0.6%</td>
<td>0.7%</td>
<td>1.5%</td>
<td>0.8%</td>
<td>3.6%</td>
</tr>
</tbody>
</table>
Accuracy – Falling Delays

<table>
<thead>
<tr>
<th>3.1 Split Simulations</th>
<th>3.2 Simplify Input Waveforms</th>
<th>3.3 Simulate Load with RICE</th>
<th>3.4 Pi Model for Driver Load</th>
<th>3.5 4D Waveform Cache</th>
<th>3.5 4D Waveform Cache (2x resolution)</th>
<th>3.6 2D Waveform Cache</th>
<th>SPICE, 50ps Maximum Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bias</td>
<td>0.0%</td>
<td>0.6%</td>
<td>0.9%</td>
<td>1.1%</td>
<td>0.1%</td>
<td>1.0%</td>
<td>-1.6%</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.1%</td>
<td>1.0%</td>
<td>1.0%</td>
<td>1.1%</td>
<td>1.6%</td>
<td>1.2%</td>
<td>3.6%</td>
</tr>
</tbody>
</table>
Accuracy – Error Distribution (4D Cache with IRICE)

Irregularity in distribution shape arises partly due to the summation of several distinct driver types into one distribution

- Worst case outliers (not shown):
  - -8.9%, +11.1% for rising delays
  - -9.0%, +15.9% for falling delays
## Runtime Profile (4D Cache with IRICE)

<table>
<thead>
<tr>
<th>Subtask</th>
<th>Delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RICE Build Circuit</td>
<td>9.3%</td>
</tr>
<tr>
<td>RICE Calculate Moments</td>
<td>36.7%</td>
</tr>
<tr>
<td>RICE Calculate Poles/Residues</td>
<td>18.0%</td>
</tr>
<tr>
<td>PWL Convolution</td>
<td>10.7%</td>
</tr>
<tr>
<td>Least Squares Fit</td>
<td>6.3%</td>
</tr>
<tr>
<td>4D Interpolation</td>
<td>4.0%</td>
</tr>
<tr>
<td>4D Cache Initialization</td>
<td>4.6%</td>
</tr>
<tr>
<td>Other</td>
<td>10.4%</td>
</tr>
</tbody>
</table>

More than 50% of runtime is spent in IRICE

- In particular, moment calculation followed by poles/residues calculation
- Outside IRICE, piecewise linear waveform convolution has the highest runtime

When compared to SPICE, overall runtime is ~3x faster at a similar accuracy level