Timing-aware routing in the RapidWright framework

Leo Liu, Nachiket Kapre
leo.liu@uwaterloo.ca, nachiket@uwaterloo.ca
Background

**RapidWright:** Open source project for accessing low-level resources for Xilinx FPGAs

**Advantage:** design generation without FPGA CAD tools

**Weakness:** no timing knowledge of FPGA resources; hard to build timing-driven tools
Problem Statement

We used RapidWright to design RapidRoute, a fast router for building communication networks.

**Problem:** RapidWright does not allow RapidRoute to be timing-driven:

- Routing algorithms cannot optimize for shortest path
- Consistently loses to Vivado
Timing Slack

![Graph showing timing slack versus ring size, torus size, and mesh size for different routing methods: RapidRoute, Vivado MoreGlobalInterations, Vivado Quick, and RapidWright. The x-axis represents the size, and the y-axis represents the worst slack in nanoseconds (ns).]
**BLUE:**
11 nodes
0.887ns

**GREEN:**
11 nodes
0.815ns
Solution

Build our own timing library

Main ideas:

- Extract relevant timing data using both RapidWright and Vivado
- Integrate extracted data into RapidRoute algorithm
Key Claim

We can extract fine-grained timing information of Xilinx FPGA routing resources

- Library allows RapidRoute to match Vivado performance
- Less than 10 mins of one-time analysis
- Extremely lightweight
  - RapidRoute retains its routing speed
  - Low memory overhead
Main Approach

1. Build many calibration designs with RapidWright
2. Load designs in Vivado for timing feedback
3. Organize into a linear system

\[
\begin{pmatrix}
1 & 2 & 5 & 0 & 0 & \ldots & 1 \\
5 & 3 & 0 & 3 & 1 & \ldots & 5 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 4 & 1 & 1 & 0 & \ldots & 0 \\
\end{pmatrix}
\begin{pmatrix}
t_1 \\
t_2 \\
\vdots \\
t_m \\
\end{pmatrix}
= 
\begin{pmatrix}
y_1 \\
y_2 \\
\vdots \\
y_m \\
\end{pmatrix}
\]

Vivado path delays (ns)
LOGIC_OUT_E19 + SNG_DBL_15 + NN1 + IMUX_7
+ BNCE_E11 + ... = 0.815ns

LOGIC_OUT_E19 + SNG_DBL_15 + NN1 + IMUX_7
+ BNCE_E11 + ... = 0.887ns

LOGIC_OUT_E19 + SNG_DBL_15 + NN1 + IMUX_7
+ BYPASS_E9 + ... = 0.756ns
Opportunities in Timing Extraction

**Symmetry:** symmetrical elements on FPGA have similar timing characteristics

**Narrow usage:** RapidRoute only targets communication overlays
Calibration Designs

Designs: **1-bit signal**, with **changing start and end nodes**

1. For each design, **route in different ways**
2. Write out each routing result into DCP
3. Track which nodes are used for each route
Experimental Setup

- Metrics:
  - Timing prediction accuracy
- Calibration designs
  - Single-bit routes of arbitrary displacement
  - Various devices and speed grades
- Compare methods:
  - Partition 70% training, 30% testing of all calibration runs
Experimental Setup

- Devices:
  - Ultrascale XCKU115 (-3, -2, -1 speed grades)
  - Ultrascale+ XCKU5P (-3, -2, -1 speed grades)
- Vivado: 2018.3
- RapidWright: 2018.3.3-beta
- Hardware: Intel Xeon E5-1630
Timing Accuracy

Measuring accuracy:

We check datapath prediction errors of 30% partition.

X-axis: size of 70% partition

Y-axis: average prediction error
Vivado Runtime
<table>
<thead>
<tr>
<th>Speed Grade</th>
<th>Logic Outs</th>
<th>Node_MUX</th>
<th>Bypass</th>
<th>Single_Dbl</th>
<th>Quad_Long</th>
<th>Inode</th>
<th>Single</th>
<th>Global</th>
<th>N1</th>
<th>N2</th>
<th>N4</th>
<th>N5</th>
<th>N12</th>
<th>S1</th>
<th>S2</th>
<th>S4</th>
<th>S5</th>
<th>S12</th>
<th>WW1</th>
<th>WW2</th>
<th>WW4</th>
<th>WW12</th>
<th>EE1</th>
<th>EE2</th>
<th>EE4</th>
<th>EE12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>89.9</td>
<td>93.5</td>
<td>97.7</td>
<td>93.6</td>
<td>44.1</td>
<td>92.5</td>
<td>55.1</td>
<td>90.1</td>
<td>59.2</td>
<td>45.2</td>
<td>49.2</td>
<td>53.2</td>
<td>36.9</td>
<td>50.9</td>
<td>40.9</td>
<td>49.9</td>
<td>43.3</td>
<td>23.0</td>
<td>27.0</td>
<td>56.0</td>
<td>37.7</td>
<td>16.0</td>
<td>22.0</td>
<td>57.0</td>
<td>30.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>89.4</td>
<td>93.3</td>
<td>97.5</td>
<td>93.4</td>
<td>44.0</td>
<td>92.3</td>
<td>55.0</td>
<td>90.0</td>
<td>59.2</td>
<td>45.1</td>
<td>49.1</td>
<td>53.1</td>
<td>36.9</td>
<td>50.8</td>
<td>40.9</td>
<td>49.8</td>
<td>43.2</td>
<td>23.0</td>
<td>26.9</td>
<td>55.0</td>
<td>37.7</td>
<td>16.0</td>
<td>22.0</td>
<td>57.0</td>
<td>30.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>94.4</td>
<td>97.9</td>
<td>54.4</td>
<td>79.9</td>
<td>59.2</td>
<td>29.3</td>
<td>79.6</td>
<td>43.3</td>
<td>42.6</td>
<td>18.4</td>
<td>29.5</td>
<td>59.1</td>
<td>68.1</td>
<td>46.9</td>
<td>20.7</td>
<td>28.7</td>
<td>56.6</td>
<td>63.2</td>
<td>49.4</td>
<td>35.7</td>
<td>38.4</td>
<td>76.4</td>
<td>50.0</td>
<td>17.7</td>
<td>23.0</td>
<td>76.8</td>
</tr>
<tr>
<td></td>
<td>100.9</td>
<td>91.6</td>
<td>41.8</td>
<td>86.2</td>
<td>54.1</td>
<td>68.4</td>
<td>73.6</td>
<td>32.4</td>
<td>38.0</td>
<td>15.1</td>
<td>29.2</td>
<td>61.0</td>
<td>73.3</td>
<td>40.4</td>
<td>23.5</td>
<td>20.0</td>
<td>76.5</td>
<td>72.3</td>
<td>37.2</td>
<td>27.1</td>
<td>42.1</td>
<td>86.3</td>
<td>54.6</td>
<td>20.1</td>
<td>57.2</td>
<td>75.6</td>
</tr>
<tr>
<td></td>
<td>139.2</td>
<td>92.4</td>
<td>67.9</td>
<td>105.1</td>
<td>86.2</td>
<td>114.2</td>
<td>116.0</td>
<td>65.7</td>
<td>62.5</td>
<td>24.1</td>
<td>48.9</td>
<td>107.8</td>
<td>115.2</td>
<td>59.9</td>
<td>35.9</td>
<td>50.8</td>
<td>114.8</td>
<td>130.1</td>
<td>67.6</td>
<td>42.5</td>
<td>42.1</td>
<td>86.3</td>
<td>54.6</td>
<td>20.1</td>
<td>57.2</td>
<td>75.6</td>
</tr>
<tr>
<td></td>
<td>179.9</td>
<td>183.9</td>
<td>105.8</td>
<td>178.2</td>
<td>113.0</td>
<td>231.4</td>
<td>176.5</td>
<td>105.5</td>
<td>147.8</td>
<td>39.6</td>
<td>113.0</td>
<td>249.3</td>
<td>131.3</td>
<td>79.4</td>
<td>42.0</td>
<td>107.2</td>
<td>213.9</td>
<td>249.9</td>
<td>97.4</td>
<td>72.7</td>
<td>38.0</td>
<td>222.1</td>
<td>141.9</td>
<td>53.8</td>
<td>99.5</td>
<td>189.0</td>
</tr>
</tbody>
</table>

Delay (ps)
Additional Notes

- Output timing database is extremely small (< 100KB)
- Majority of timing extraction solver runtime is due to Vivado query wait times
Integrating with RapidRoute

- RapidRoute accepts timing database file(s) as input, overwriting default heuristic
- Heuristic has nearly identical computing cost as default heuristic
Experimental Setup

● Metrics:
  ○ Timing performance
  ○ Routing runtimes

● Communication structures:
  ○ 1D rings, 2D torii, 2D meshes

● Compare methods:
  ○ RapidRoute default, RapidRoute+Timing, Vivado
Experimental Setup

- Device: Ultrascale XCKU115 xcku115-flva1517-3-e
- Vivado: 2018.3
- RapidWright: 2018.3.3-beta
- Hardware: 32-core 2.6GHz Intel Xeon
Timing Results
Routing Runtime

![Graph showing execution time for different routing algorithms across varying network sizes.]

- **RapidRoute**
- **Vivado MoreGlobalIterations**
- **Vivado Quick**
- **RapidWright**

The graphs illustrate execution time (in seconds) for different network topologies (Ring size, Torus size, Mesh size) and routing algorithms, highlighting performance trends and comparisons.
Conclusion

- We developed a **timing extraction tool**, which is **light-weight** and **highly-accurate**
- Timing results expected to be within 1% error margin
- Total calibration phase takes minutes
- Extremely lightweight output and usage
Improved RapidRoute

- RapidRoute retains a **5-8x** routing speed advantage over Vivado
- RapidRoute now gains **competitive timing performance** on communication overlay designs