Fast Linking of Separately-Compiled FPGA Blocks without a NoC

Yuanlong Xiao, Syed Tousif Ahmed, and André DeHon
ylxiao,stahmed@seas.upenn.edu, andre@ieee.org

Implementation of Computation Group
University of Pennsylvania
December 12th, 2019
Fast Linking of Separately-Compiled FPGA Blocks without a NoC

Content:

- Motivation
- Approach
- Evaluation
- Future Work
- Conclusion
Motivation

● Today’s FPGA compilation is slow
    ■ Compiled on Google Cloud Computing Engines.
    ■ 4 Dual-thread, 2.8GHz Intel Xeon Cascade Lake Processors
    ■ 64GB RAM
    ■ SDSoC runs with 8 threads

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit Recognition</td>
<td>41.2 min</td>
</tr>
<tr>
<td>SPAM Filter</td>
<td>29.5 min</td>
</tr>
<tr>
<td>3-D Rendering</td>
<td>29.4 min</td>
</tr>
<tr>
<td>Optical Flow</td>
<td>44.3 min</td>
</tr>
<tr>
<td>Binarized Neural Network</td>
<td>178 min</td>
</tr>
<tr>
<td>Face Detection</td>
<td>71.9 min</td>
</tr>
</tbody>
</table>

Motivation

- PRFlow[2] can reduce compile time to 10-16 minutes
  - Big Idea: Divide-and-conquer
  - Pre-compile Overlay & Compile Separate Blocks in Parallel
  - 10-16 minutes Compile Time

\[ T_{new} = \frac{T_{origin}}{\# \text{ of Partition}} \]

## Motivation

- PRFlow[2] can reduce compile time to 10-16 minutes
  - Big Idea: Divide-and-conquer
  - Pre-compile Overlay & Compile Separate Blocks in Parallel
  - 10-16 minutes Compile Time

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit Recognition</td>
<td>41.2 min</td>
<td>10.6 min</td>
</tr>
<tr>
<td>SPAM Filter</td>
<td>29.5 min</td>
<td>10.9 min</td>
</tr>
<tr>
<td>3-D Rendering</td>
<td>29.4 min</td>
<td>10.9 min</td>
</tr>
<tr>
<td>Optical Flow</td>
<td>44.3 min</td>
<td>12.4 min</td>
</tr>
<tr>
<td>Binarized Neural Network</td>
<td>178 min</td>
<td>16.4 min</td>
</tr>
<tr>
<td>Face Detection</td>
<td>71.9 min</td>
<td>16.2 min</td>
</tr>
</tbody>
</table>

Motivation

- Performance Degradation
  - Slow down by 1.3-12.4 times compared with SDSoc version

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit Recognition</td>
<td>13.00 ms</td>
<td>16.58 ms (1.3X)</td>
</tr>
<tr>
<td>SPAM Filter</td>
<td>82.13 ms</td>
<td>48.90 ms (0.5X)</td>
</tr>
<tr>
<td>3-D Rendering</td>
<td>6.17 ms</td>
<td>1.18 ms (0.19X)</td>
</tr>
<tr>
<td>Optical Flow</td>
<td>6.35 ms</td>
<td>25.80 ms (4.1X)</td>
</tr>
<tr>
<td>Binarized Neural Network</td>
<td>5.30 ms</td>
<td>17.42 ms (3.3X)</td>
</tr>
<tr>
<td>Face Detection</td>
<td>28.19 ms</td>
<td>351.93 ms (12.4X)</td>
</tr>
</tbody>
</table>

Motivation

- Today’s FPGA compilation is slow: **30-178 minutes**
- PRFlow[2] can reduce compile time to **10-16 minutes**
- PRFlow[2] can degrade performance by **1.2-12.5 times**

**We propose:**
- Keep Separate Compilations
- Exploit More FPGA bandwidth
Motivation: Packet-Switched Network-on-Chip (PSNoC) IO Bottleneck

- 1 cycle for computing
- 1 cycle for read/write per link
Motivation: Packet-Switched Network-on-Chip (PSNoC) IO Bottleneck
Motivation: Packet-Switched Network-on-Chip (PSNoC) IO Bottleneck

- Bandwidth Limits in PSNoC
  - Packet-Switched Network does not Use all Raw Wires
  - Sequentializing Multiplexers share 9.6 gbps ($32$bits*300MHz) throughput across the Boundary
Motivation: Benchmark[2] Profile on PSNoC

We need more Bandwidth!

\[
\text{NormComputeCycles} = \frac{\text{ComputeCycles}}{\text{Max}\{\text{AllComputeCycle}\}}
\]

\[
\text{NormIoCycles} = \frac{\text{IoCycles}}{\text{Max}\{\text{AllComputeCycle}\}}
\]

Why not increase the PSNoC datawidth?

32 bits PSNoC Occupies 20% LUTs

Only limited number of links need more bandwidth

Motivation: Packet-Switched Network-on-Chip (PSNoC) IO Bottleneck

- Increase bandwidth
- Not much unnecessary routing overhead?
Approaches: Direct Wires

- Pre-route as many wires as possible
  - Use all Raw Wires
  - >> 9.6 gbps throughput across the Boundary
  - Switch box and pages are reconfigurable
  - Compiled in parallel
What we accomplish:

- Keep Separate Compilations
- Exploit More FPGA bandwidth

Approaches: Direct Wires

No Bandwidth Bottleneck
Approaches: Bandwidth Upper Bound

We have more wires!
- Max number of Wire cross the boundary
- Maximum: 86.4Gbps (288X300MHz)

TABLE II: Static Timing Analysis Slack for 60 CLB Boundary

<table>
<thead>
<tr>
<th>W</th>
<th>100</th>
<th>200</th>
<th>250</th>
<th>300</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>3.761</td>
<td>0.861</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>64</td>
<td>1.187</td>
<td>0.284</td>
<td>0.287</td>
<td>X</td>
<td>-0.853</td>
</tr>
<tr>
<td>96</td>
<td>3.151</td>
<td>0.438</td>
<td>-0.764</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>128</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.021</td>
<td>-1.009</td>
</tr>
<tr>
<td>160</td>
<td>3.292</td>
<td>0.536</td>
<td>X</td>
<td>X</td>
<td>-0.905</td>
</tr>
<tr>
<td>192</td>
<td>3.262</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>224</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-0.01</td>
<td>X</td>
</tr>
<tr>
<td>256</td>
<td>X</td>
<td>0.431</td>
<td>0.071</td>
<td>0.002</td>
<td>X</td>
</tr>
<tr>
<td>288</td>
<td>3.262</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.009</td>
</tr>
</tbody>
</table>

Numbers in cells represent the slack; slack less than zero fail to meet the timing target. X means the routing cannot be completed.
Approaches: A further step

We have more Sides!

- Pre-route wires between user pages
- Proper placement to use locality.
Approach

- **Point-to-point Wires (Direct Wires)**
  - FPGA has more wires than we have already used for PSNoC

- **Compile Separate FPGA Blocks with Divide-and-Conquer**
  - Define switch box as reconfigurable
  - Fast Compilation

![Diagram of DMA_in and DMA_out with nodes a, b, c, d, e and switch box with labels a, b, c, d, empty]
Evaluation

- Compile servers: Google Cloud
  - Each compute node with 4 dual-thread, 2.8GHz Intel Xeon Cascade Lake Processors
- Platform for PRflow on Vivado 2018.2
  - Xilinx ZCU102 board with xczu9eg-ffvb1155-2-e MP-SOC chip
  - 274K LUTs, 912 BRAM36, 2520 DSPs
  - 775 MHz clock for Fabric
- Rosetta HLS Benchmark [1]
  - 6 C-based design for High Level Synthesis Benchmark
  - 3-D Rendering, Digit-Recognition, Spam-filter, Optical-flow, BNN, Face-detection
  - We partitioned the benchmarks into small pieces, details in the paper

Evaluation: PSNoC Layout

- 20 leaves for application logic
- 1 leaf for 4-core ARM processor
- 1 leaf for DMA interface
- 20 leaves are connected by BFT
Evaluation: DW Layout

- 14 user pages, 8 switch pages
- User logic can also borrow resource from switch pages
- 1 page for 4-core ARM processor
- 1 page for DMA interface
- Wires numbers vary from 130 to 424
Evaluation: Benchmark Mapping

- Mapping strategy for all the benchmarks

(a) Rendering

(b) Digit Recognition

(c) BNN

(d) Spam Filter

(e) Optical Flow

(f) Face Detection

Legend:
- Mapped User Page
- Mapped Mixed Page
- Pure Switch Box Page
- Empty User Page
Evaluation: Compile Time

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rendering</td>
<td>1711</td>
<td>1495</td>
<td>606</td>
<td>737</td>
</tr>
<tr>
<td>Digit Recognition</td>
<td>2569</td>
<td>2104</td>
<td>610</td>
<td>735</td>
</tr>
<tr>
<td>Spam Filter</td>
<td>1930</td>
<td>1780</td>
<td>568</td>
<td>695</td>
</tr>
<tr>
<td>Optical Flow</td>
<td>2997</td>
<td>2792</td>
<td>679</td>
<td>886</td>
</tr>
<tr>
<td>Binarized Neural Network</td>
<td>12001</td>
<td>11089</td>
<td>1004</td>
<td>1082</td>
</tr>
<tr>
<td>Face Detection</td>
<td>4136</td>
<td>2981</td>
<td>825</td>
<td>1089</td>
</tr>
</tbody>
</table>

Unit: second

- Our user logic compile time is similar to PSNoC[2] Compile time
- The switch box does not increase the compile time
- Digit Recognition has systolic array computing graph, so it does not need to use switch box.

## Evaluation: Performance Improvements

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rendering</td>
<td>1.5</td>
<td>1.4</td>
<td>1.2</td>
<td>1.3 (0.9X)</td>
</tr>
<tr>
<td>Digit Recognition</td>
<td>6.9</td>
<td>5.0</td>
<td>10.8</td>
<td>5.4 (2.0X)</td>
</tr>
<tr>
<td>Spam Filter</td>
<td>28.2</td>
<td>22.4</td>
<td>48.9</td>
<td>32.2 (1.5X)</td>
</tr>
<tr>
<td>Optical Flow</td>
<td>3.5</td>
<td>2.1</td>
<td>25.8</td>
<td>2.6 (9.9X)</td>
</tr>
<tr>
<td>Face Detection</td>
<td>18.2</td>
<td>24.3</td>
<td>101.0</td>
<td>33.1 (3.1X)</td>
</tr>
<tr>
<td>BNN</td>
<td>5.3</td>
<td>3.6</td>
<td>17.4</td>
<td>22.4 (0.8X)</td>
</tr>
<tr>
<td>BNN_new</td>
<td>5.3</td>
<td>3.6</td>
<td>17.4</td>
<td>5.7 (3.0)</td>
</tr>
</tbody>
</table>

- We refactor the code for PSNoC, and improve the performance of Digit Recognition, and Face detection.
- Compared with the improved PSNoC, DW can improve the performance by 1.5x to 9.9x.
- **BNN**
  - User can run @250MHz for PSNoC
  - User can only run @187MHz for DW
- **BNN new** (After this paper)
  - Add 2 pipeline stages
  - Increase the Compile Time from 1082s to 1458s

---

Evaluation: Throughput vs. Compile Time

- Today’s FPGA compilation is slow: **30-178 minutes**
- PRFlow[2] can reduce compile time to **10-16 minutes**, but degrade the performance at most **12.5 times**
- DW can keep compile time to **12-18 minutes**, improve the performance by at most **10 times**

We propose:
- Keep Separate Compilations
- Exploit More FPGA bandwidth
Future work:

- User Page and switch box page assignment automation
- PSNoC and DW hybrid overlay for scalability
Conclusion

- Improve the Application performance by at most 10X
- Able to keep the compile time within 18 minutes
- Decrease the interface area by 47-86%.
Thank you

Q&A