Intermediate Fabrics: Virtual Architectures for Mainstream Reconfigurable Computing

This work is supported by National Science Foundation grant CNS-1149285 and the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422.
Introduction

- Goal: enable FPGA usage by designers currently targeting GPUs and multi-cores
- Problem: order-of-magnitude worse productivity
- Productivity bottlenecks
  - Register-transfer-level (RTL) design
    - Solved by high-level synthesis (HLS)
  - Long compile times (hours to days)
    - Prevents mainstream methodologies
  - Limited portability
    - Limits design reuse
- Solution: Intermediate Fabrics (IFs)
  - Virtual, reconfigurable architectures between application and FPGA
  - Provides near-instant FPGA compilation via abstraction
    - 700x-4000x faster than commercial tools
  - Integrates with OpenCL HLS to enable transparent FPGA usage

```c
__kernel void kernelA(int *data) { ... }
```

---

Fast IF reconfiguration enables support for multiple kernels and rapid coding changes.

FPGA reconfiguration loads new fabrics to support new app requirements.

Fast compilation due to small input size as opposed to 100k+ LUTs.

Intermediate Fabric (IF) w/ Floating-Point Resources.
Intermediate Fabric (IF) Overview

Traditional FPGA Tool Flow

- Synthesis
  - > 10k lookup-tables (LUTS)
- Place & Route (PAR)
  - Lengthy compilation
- Bitfile
  - FPGA specific: Not portable
- FPGA

Intermediate Fabric Tool Flow

- Synthesis, Place & Route
- Intermediate Fabric (IF) w/ Floating-Point Resources
  - Fast Compilation: several coarse-grained resources
  - App Portability: always targets IF regardless of underlying FPGA
  - Fast Partial Reconfiguration: even on devices without support

Fabric Library

Virtual Device

Physical Device(s)
Research Challenges

- How to minimize area/performance overhead?

- How to customize fabric overlay for different FPGA fabrics?
  - CASES 2012, ASAP 2013

- How to specialize fabric for a set of applications?
  - IEEE Micro (to appear)

- How to integrate with OpenCL high-level synthesis?
  - IEEE Micro (to appear)
Intermediate Fabric (IF) Architecture

- Fabric can implement *any* architecture
  - Current focus on island-style layout
    - Switch boxes, connection boxes, tracks
  - App-specialized computational units (CUs)
    - FFTs, floating-point resources, filters, etc.
  - Specialized track widths

### Island-Style Layout

#### Virtual Track

- **Switch Box West**
- **Routing Track**
- **Connection Box**
- **CU North**
- **Output**
- **Switch Box East**
- **Routing Track**
- **CU South**
- **Input**

### "Soft" RTL Track Implementation

- **Switch Box (SB)**
- **Connection Box (CB)**
- **Computational Unit (CU)**
- **Tracks**
- **Switch Box (SB)**
- **Connection Box (CB)**

For a *n*-bit track with *m* sources, circuit uses a *m*: 1, *n*-bit mux

Many tracks in IF, largest source of overhead

- Switch boxes implemented similarly
  - Mux defines every connection
  - Supports any topology
  - Specialized to application requirements

- Optional registers on outputs
  - Eliminates combinational loops
  - Minimizes delays across muxes

- Pipelined interconnect can require complicated routing
  - Ensures routing paths have same # of hops

- For pipelined circuits, avoid by using realignment registers
  - Lengthens shorter path, adds pipeline stages
  - Enables use of traditional place & route algorithms

CHREC
NSF Center for High-Performance Reconfigurable Computing
## Preliminary Experimental Results

<table>
<thead>
<tr>
<th>Place and Route Times</th>
<th>Performance</th>
<th>Area Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IF Quartus 9.1 Speedup</td>
<td>Clk FPGA</td>
</tr>
<tr>
<td>Conv 3x3</td>
<td>0.9s 14min 48s 943</td>
<td>150 MHz</td>
</tr>
<tr>
<td>Conv 4x4</td>
<td>1.5s 15min 06s 613</td>
<td>148 MHz</td>
</tr>
<tr>
<td>Conv 5x5</td>
<td>2.1s 15min 33s 447</td>
<td>146 MHz</td>
</tr>
<tr>
<td>Conv 6x6</td>
<td>3.0s 15min 41s 312</td>
<td>151 MHz</td>
</tr>
<tr>
<td>Conv 7x7</td>
<td>4.0s 16min 19s 243</td>
<td>139 MHz</td>
</tr>
<tr>
<td>Conv 8x8</td>
<td>5.3s 16min 08s 184</td>
<td>146 MHz</td>
</tr>
<tr>
<td>Sobel</td>
<td>4.2s 14min 56s 214</td>
<td>154 MHz</td>
</tr>
<tr>
<td>SAD 8x8</td>
<td>5.3s 16min 51s 190</td>
<td>143 MHz</td>
</tr>
<tr>
<td>Conv 5x5 (float)</td>
<td>1.7s 25min 28s 919</td>
<td>148 MHz</td>
</tr>
<tr>
<td>Sobel (float)</td>
<td>1.5s 18min 58s 759</td>
<td>144 MHz</td>
</tr>
<tr>
<td>SAD 5x5 (float)</td>
<td>0.6s 30min 43s 2880</td>
<td>140 MHz</td>
</tr>
<tr>
<td>Average</td>
<td>2.7s 18min 14s 700</td>
<td>146 MHz</td>
</tr>
</tbody>
</table>

- **Avg. place & route speedup:** 700x
- **Avg. performance overhead:** 7%
- **IF used 2.2x to 4.4x more LUTs than FPGA circuits**
  - However, IF pessimistically implemented with only soft routing resources
  - IF also capable of implementing numerous circuits with fast partial reconfiguration (28-72 cycles)
  - Compared to an FPGA circuit with 3-4 kernels, IF saves area (assuming kernels are not concurrent)
Optimizing Interconnect for FPGAs

- Almost all overhead due to muxes
- However, mux LUT utilization plateaus as number of inputs grows
  - Optimization opportunity
- Switchboxes previously used 3-input muxes
  - Can add one input for free
  - Improves routability
  - Reduce routing resources and overhead
- ~50% reduction in LUTs

![Graph showing the number of 4-input LUTs vs. number of Mux Inputs for different data widths (16-bit, 32-bit, 64-bit)]
**Main idea:** automatic design of intermediate fabrics for multi-kernel applications

@ system generation, clustering heuristic groups app’s kernels based on resource requirements
- Custom reconfiguration context IF is created for each group

@ kernel compile time (potentially @ runtime), kernel mapped quickly onto pre-existing context
- IF flexibility allows supporting similar **new** or **changed** kernels w/o hardware regeneration

@ runtime, reconfigure the device with compatible context if not already resident
- Because only IF operators are fixed, new kernels may already be compatible with existing contexts
- In worst case, create new IF to support changes
OpenCL-IF Case Study

- Evaluated computer vision system with 10 fixed-/floating-point OpenCL kernels
- Compared OpenCL-IF compile times and area/performance against VHDL
- System of kernels compile in ~6.5s total vs. 7.4h direct: 4211x system speedup
- 4x faster for FLT vs. FXD

### Table: Reconfiguration Contexts

<table>
<thead>
<tr>
<th>Per-Kernel Average/Total</th>
<th>OpenCL-IF Time</th>
<th>PAR Time</th>
<th>Total Time</th>
<th>Clock</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR, Gaussian, Sobel</td>
<td>0.123s</td>
<td>0.459s</td>
<td>0.582s</td>
<td>225MHz</td>
<td>13.4%</td>
</tr>
<tr>
<td><strong>Cluster 1 total</strong></td>
<td>0.369s</td>
<td>1.377s</td>
<td>1.746s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bilinear, Mean, Threshold</td>
<td>0.102s</td>
<td>0.177s</td>
<td>0.279s</td>
<td>256MHz</td>
<td>10.8%</td>
</tr>
<tr>
<td><strong>Cluster 2 total</strong></td>
<td>0.306s</td>
<td>0.532s</td>
<td>0.838s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max, Min, Normalize, SAD</td>
<td>0.165s</td>
<td>0.096s</td>
<td>0.261s</td>
<td>225MHz</td>
<td>16.2%</td>
</tr>
<tr>
<td><strong>Cluster 3 total</strong></td>
<td>0.661s</td>
<td>0.383s</td>
<td>1.044s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIR, Gaussian FLT</td>
<td>0.115s</td>
<td>0.237s</td>
<td>0.353s</td>
<td>196MHz</td>
<td>40.4%</td>
</tr>
<tr>
<td><strong>Cluster 4 total</strong></td>
<td>0.346s</td>
<td>0.712s</td>
<td>1.059s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bilinear, Mean, Threshold, Max, Min, Normalize, SAD FLT</td>
<td>0.137s</td>
<td>0.108s</td>
<td>0.245s</td>
<td>196MHz</td>
<td>48.0%</td>
</tr>
<tr>
<td><strong>Cluster 5 total</strong></td>
<td>0.959s</td>
<td>0.753s</td>
<td>1.712s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel average</td>
<td>0.132s</td>
<td>0.188s</td>
<td>0.320s</td>
<td></td>
<td>25.8%</td>
</tr>
<tr>
<td>System total</td>
<td>2.642s</td>
<td>3.758s</td>
<td>6.399s</td>
<td>128.8%</td>
<td></td>
</tr>
</tbody>
</table>

### Table: Direct FPGA Implementations

<table>
<thead>
<tr>
<th>Per-Kernel Average/Total</th>
<th>XST Time</th>
<th>PAR Time</th>
<th>Total Time</th>
<th>Clock</th>
<th>Compilation Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR, Gaussian, Sobel</td>
<td>10.8s</td>
<td>85s</td>
<td>96s</td>
<td>275MHz</td>
<td>166x</td>
</tr>
<tr>
<td><strong>Cluster 1 total</strong></td>
<td>32.4s</td>
<td>255s</td>
<td>288s</td>
<td></td>
<td>165x</td>
</tr>
<tr>
<td>Bilinear, Mean, Threshold</td>
<td>9.4s</td>
<td>1110s</td>
<td>1120s</td>
<td>292MHz</td>
<td>3,893x</td>
</tr>
<tr>
<td><strong>Cluster 2 total</strong></td>
<td>28.2s</td>
<td>3331s</td>
<td>3359s</td>
<td></td>
<td>4,007x</td>
</tr>
<tr>
<td>Max, Min, Normalize, SAD</td>
<td>11.1s</td>
<td>380s</td>
<td>391s</td>
<td>236MHz</td>
<td>1,350x</td>
</tr>
<tr>
<td><strong>Cluster 3 total</strong></td>
<td>44.3s</td>
<td>1520s</td>
<td>1565s</td>
<td></td>
<td>1,499x</td>
</tr>
<tr>
<td>FIR, Gaussian FLT</td>
<td>9.4s</td>
<td>4790s</td>
<td>4800s</td>
<td>132MHz</td>
<td>14,534x</td>
</tr>
<tr>
<td><strong>Cluster 4 total</strong></td>
<td>28.3s</td>
<td>14371s</td>
<td>14400s</td>
<td></td>
<td>13,603x</td>
</tr>
<tr>
<td>Bilinear, Mean, Threshold, Max, Min, Normalize, SAD FLT</td>
<td>10.0s</td>
<td>1038s</td>
<td>1048s</td>
<td>199MHz</td>
<td>3,734x</td>
</tr>
<tr>
<td><strong>Cluster 5 total</strong></td>
<td>69.7s</td>
<td>7265s</td>
<td>7335s</td>
<td></td>
<td>4,865x</td>
</tr>
<tr>
<td>Kernel average</td>
<td>10.2s</td>
<td>1337s</td>
<td>1347s</td>
<td>222MHz</td>
<td>4,366x</td>
</tr>
<tr>
<td>System total</td>
<td>203.0s</td>
<td>26742s</td>
<td>26947s</td>
<td></td>
<td>13,603x</td>
</tr>
</tbody>
</table>

### Table: Overhead

<table>
<thead>
<tr>
<th>Per-Kernel Average/Total</th>
<th>Clock</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR, Gaussian, Sobel</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cluster 1 total</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bilinear, Mean, Threshold</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cluster 2 total</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max, Min, Normalize, SAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cluster 3 total</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIR, Gaussian FLT</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cluster 4 total</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bilinear, Mean, Threshold, Max, Min, Normalize, SAD FLT</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cluster 5 total</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel average</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System total</td>
<td></td>
<td>1.8x</td>
</tr>
</tbody>
</table>

0.32s compile times demonstrate feasibility of runtime kernel compilation

1.8x system area overhead, 1.3x-15x per context vs. separate accelerators

- Overhead amortized over multiple kernels by using the IF’s rapid configurability
- Effective overhead decreases w/ new kernels!
- Lower for FLT due to larger ops

---

Xilinx ISE 14.4 using reduced effort for faster compilation at expense of circuit quality for XC6VCX130T-1FF1154. Times on quad-core 2.66 GHz Intel Xeon W3520 workstation with 12GB RAM running CentOS 6.4 x86 64.
Conclusions and Future Work

- Intermediate fabrics enable near-instant (< 1s) FPGA compilation
  - 700x-4000x faster than device-vendor tools
- Performance overhead is modest (~7%)
- Area overhead can be significant for some use cases
  - Significant focus of ongoing work

Future work

- Novel interconnect architectures to reduce area overhead
- High-level synthesis optimizations enabled by fast compilation
- Partial reconfiguration of fabric resources

References