Cross-Platform FPGA Accelerator Development Using CoRAM and CONNECT

Eric S. Chung, Michael K. Papamichael, Gabriel Weisz, James C. Hoe, Ken Mai

Microsoft Research

Carnegie Mellon
USB drives are being passed around

Please copy all files

and install VirtualBox
Start the Virtual Machine

• Make sure VirtualBox 4.2.4 is installed
• Double-click “CoRAM Demo.ova”
• Click “Import”
• Double-click “CoRAM Demo” to start the VM
• Log into the VM with username “lubuntu” and password “lubuntu”
Today’s Agenda

• Main Course
  – 9:30—10:00: CoRAM Overview
  – 10:00—10:45: Hands-On Exercise

• Coffee
  – 10:45—11:00

• Desserts
  – 11:00—11:30: CONNECT & CoRAM: Beneath the Hood
  – 11:30—12:30: Advanced Topics and Exercises
CoRAM: A Portable and Scalable Memory Architecture for FPGA-Based Computing

www.ece.cmu.edu/~coram
Where Is The Future Headed?

2009 Intl. Technology Roadmap for Semiconductors

- Area density
- Moore’s Law
- 16X area density

Normalized to 40nm (log)

Technology Node (nm)

Normalized area density increases over time, according to Moore’s Law, with a significant leap in 2022.
Where Is The Future Headed?

2009 Intl. Technology Roadmap for Semiconductors

- Area density
- Supply voltage
- Power (device)

Normalized to 40nm (log)

- 16X area density
- 4X lower power density

Moore’s Law

Must do more Ops/sec for less Joules/sec
The Energy Efficiency of General Purpose Processors

Over 90% energy of general-purpose von Neumann processor is “overhead”
What Do With 10B Transistors in 2020?


## Efficiency of Specialized HW [MICRO’10]

<table>
<thead>
<tr>
<th>Device</th>
<th>GFLOP/s actual</th>
<th>(GFLOP/s)/mm(^2) normalized to 40nm</th>
<th>GFLOP/J normalized to 40nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i7 (45nm)</td>
<td>96</td>
<td>0.50</td>
<td>1.14</td>
</tr>
<tr>
<td>Nvidia GTX285 (55nm)</td>
<td>425</td>
<td>2.40</td>
<td>6.78</td>
</tr>
<tr>
<td>Nvidia GTX480 (40nm)</td>
<td>541</td>
<td>1.28</td>
<td>3.52</td>
</tr>
<tr>
<td>ATI R5870 (40nm)</td>
<td>1491</td>
<td>5.95</td>
<td>9.87</td>
</tr>
<tr>
<td>Xilinx V6-LX760 (40nm)</td>
<td>204</td>
<td>0.53</td>
<td>3.62</td>
</tr>
<tr>
<td>same RTL std cell (65nm)</td>
<td>---</td>
<td>19.28</td>
<td>50.73</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>GFLOP/s</th>
<th>(GFLOP/s)/mm(^2)</th>
<th>GFLOP/J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i7 (45nm)</td>
<td>67</td>
<td>0.35</td>
<td>0.71</td>
</tr>
<tr>
<td>Nvidia GTX285 (55nm)</td>
<td>250</td>
<td>1.41</td>
<td>4.2</td>
</tr>
<tr>
<td>Nvidia GTX480 (40nm)</td>
<td>453</td>
<td>1.08</td>
<td>4.3</td>
</tr>
<tr>
<td>ATI R5870 (40nm)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Xilinx V6-LX760 (40nm)</td>
<td>380</td>
<td>0.99</td>
<td>6.5</td>
</tr>
<tr>
<td>same RTL std cell (65nm)</td>
<td>952</td>
<td>239</td>
<td>90</td>
</tr>
</tbody>
</table>
So Why Doesn’t Everybody Want To Compute With FPGAs?
“Traditionally, FPGAs have been the bastard step-brother of ASICs...”

Andre Déhon, Proceedings of ISFPGA 2004
Review of FPGA Anatomy

I/O Interfaces

Programmable lookup tables (LUT)

1000s of tiny SRAMs

4kB SRAM

Interconnect

FPGA

2/11/2013

CoRAM and CONNECT Tutorial / FPGA'13
FPGAs on the Memory Bus

**Shared Memory Multiprocessor**

- Multicore
- Multicore
- FPGA
- Memory Interconnect
- DRAM
- DRAM
- DRAM

**Single-Chip Multicore**

- Core
- Core
- FPGA Fabric
- Caches
- Caches
- On-Chip Network
- DRAM Controllers
FPGA “Computers” Today

I/O Interfaces

<table>
<thead>
<tr>
<th></th>
<th>Memory Controller</th>
<th>Memory Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I/O Interfaces

Memory Controller

Memory Controller

...
FPGA “Computers” Today

I/O Interfaces

Memory Controller

Memory Controller

Application

SRAM

Control Logic

To/From Network

Memory Controller

Memory Controller

I/O Interfaces

2/11/2013

CoRAM and CONNECT Tutorial / FPGA'13
No Portability Either

Convey Computer

Xilinx ML605

Nallatech ACP
Connected RAM FPGA [FPGA’11]
Connected RAM FPGA [FPGA’11]

Memory Interfaces (Global Address Space)

General-purpose Network-on-Chip

Core Logic

CoRAM

Software control threads

General-purpose Network-on-Chip

Memory Interfaces (Global Address Space)
Connected RAM FPGA [FPGA’11]

User’s View of FPGA
Matrix-Matrix Multiply

Control Thread Is High-Level Description of Application’s Memory Access Pattern
What “Runs” CoRAM?

FPGA

Control Logic

Fabric

Cluster DMA

Network-on-Chip

Memory Translation (TLBs)

Memory Interfaces and Caches

Control thread programs

Architecture

Microarchitecture
Summary

• Future Must Look Beyond General Purpose
  – power is first-class constraint in future systems
  – FPGAs offer impressive performance and efficiency
    but should be easy-to-use and portable

• CoRAM Memory Architecture
  – high-level abstraction for memory management
  – simplifies H/W design and enables portability
The Rest of This Tutorial

• Learn how to write CoRAM applications
• Compile and simulate your H/W designs
• CoRAM: Beneath the Hood
• FPGA Demo
CoRAM: “The Experience”
Hands-On Overview

• You will build 2 applications using CoRAM:
  – “Hello World”
  – FFT Accelerator

• You will compile and run the applications in an RTL simulator
  – everything is synthesizable
  – demo on real hardware later
Before we begin

• Prerequisites
  – Familiarity with Verilog and C is useful
  – Laptop w/ 15GB of free space
  – VirtualBox Installed

• USB sticks should be going around
  – Please copy everything to a scratch folder
Start the Virtual Machine

• Make sure VirtualBox 4.2.4 is installed
• Double-click “CoRAM Demo.ova”
• Click “Import”
• Double-click “CoRAM Demo” to start the VM
• Log into the VM with username “lubuntu” and password “lubuntu”
Lesson 1: Hello World
Lesson 1: Hello World

• Double-click the “LXTerminal” icon on desktop

• In the shell, type:
  
  $ cd ~/coram/proj/HelloWorld

• Open the two main source files:
  
  $ emacs HelloWorld.c &
  $ emacs HelloWorld.v &

  (ignore HelloWorld.spec for now)
Lesson 1: Hello World

Control Thread (HelloWorld.c)

1) Perform Blocking Memory Read
2) Send token
3) Print Message

Channel FIFO

Core Logic (HelloWorld.v)

1. Read token from thread
2. Print data from CoRAM

Data transferred

Memory

169x481
Hello World Control Thread

cpi_register_thread("ctrl_thread", 1);
cpi_hand ram = cpi_get_rams(1, false, 0, 0);
cpi_hand cfifo = cpi_get_channel(cpi_fifo, 0);
cpi_write_ram(ram, 0x0, 0x0, 4);
cpi_write_channel(cfifo, 0x1);
cpi_printf("[Ctrl-Thread] Finished with all memory transfers\n");
How To Instantiate a CORAM (Verilog)

CoRAM Parameters

```
CORAM#("ctrl_thread", 0, 0, 32, 1024, 0)
Coram(
    .RST_N( RST_N ),
    .CLK  ( CLK   ),
    .en   ( … ),
    .wen  ( … ),
    .addr ( … ),
    .din  ( … ),
    .dout ( … )
)
```

Library “Black-Box” Module

Instance name

CoRAM and CONNECT Tutorial / FPGA'13
Open HelloWorld.v

• Hardware Structures
  – CoRAM is used as “portal” to DRAM
  – Channel FIFO
    • used for direct communication w/ thread
    • 1-bit “din_rdy” signal means a token is waiting

• Simulation Processes
  – Initial process prints message after reset
  – Sequential process waits for the token, prints a message, and stops the simulation
Build and run the simulator

• Commands:
  \[
  \begin{align*}
  & \$ \text{ cd } ~/\text{coram} \\
  & \$ \text{ make HelloWorld-corflow} \\
  & \$ \text{ cd HelloWorld-build} \\
  & \$ \text{ ./run_icarus.sh}
  \end{align*}
  \]

• Explanation
  – “make” compiles application to RTL + testbench
  – “run_icarus” runs RTL simulation
$ Starting simulation
$ [Core-Logic] RST_N: 0 @ Time: 0
$ Lowering reset
$ [Core-Logic] RST_N: 1 @ Time: 101
$ [Ctrl-Thread] Finished with all memory transfers
$ [Core-Logic] Received token @ Time:192 and data: 0xdeadbeef

Data is in ASCII memory image file @
~/coram/ramdata/HelloWorld.hex

(More on how pre-loading works later)
Lesson 2: FFT Accelerator
Streaming 8-point DFT Core

\[
\begin{array}{c}
\text{X0} \quad 32 \\
\text{X1} \quad 32 \\
\text{X2} \quad 32 \\
\text{X3} \quad 32 \\
\text{next} \quad \text{next\_out}
\end{array}
\]
Generate FFT-8 Core

Generate FFT Core Using This Link:
http://spiral.net/hardware/dftgen.html
(Use Chromium on the desktop – the link is in the bookmarks bar)

Set Transform Size = 8
Data Type = Fixed-Point, 32 bits
Streaming Width = 2
Leave the rest at defaults
DFT Core Plus CoRAM (top.v)

CoRAM 128

DFT-8

CoRAM 128

Address Counter

Software Control Thread

core_start  last_addr

core_done

Address Counter
Open DFT Project Files

• Commands:

$ “cd coram/proj/DftDemo”
$ emacs DftDemo.c
$ emacs DftDemo.v &
$ emacs spiral_dft.v &
void dft_thread()
{
    cpi_register_thread("dft_thread", 1);
    cpi_hand in_ram = cpi_get_rams(1, false, 0, 0);
    cpi_hand out_ram = cpi_get_rams(1, false, 1, 0);
    cpi_hand cfifo = cpi_get_channel(cpi_fifo, 0, 0);
    
    // Read from memory (8 samples x 2 complex x 4B = 64B)
    cpi_write_ram(in_ram, RAM_ADDR, IN_ADDR, 64);

    // Send "core_start" signal to FFT core
    cpi_write_channel(cfifo, 0);

    // Wait for "last_addr" from FFT core
    cpi_read_channel(cfifo);

    // Write output to memory (64B)
    cpi_read_ram(out_ram, RAM_ADDR, OUT_ADDR, 64);
}
DFT Core Logic

• One read CoRAM and one write CoRAM
• Address counters for read and write CoRAMs
  – Minor parameter difference for delay on write
• Sequential block with debug output
• Makefile ensures DFT core is copied properly
Memory Image

• Image file is ~/coram/ramdata/dftdemo.hex
  – image has more data than demo uses

• Data is hexadecimal ASCII format
  – each line contains 64 bytes
  – least significant byte is on the right
  – mimics DRAM interface

• Data layout is application specific
  – this application stores 4 bytes big endian integers
Build and Run the simulator

• Type the following commands on shell:
  
  $ cd ~/coram
  $ make DftDemo-corflow
  $ cd DftDemo-build
  $ ./run_icarus.sh

• You should see traces on the command-line
Expected Simulation Output

Starting simulation
Lowering reset
188: CORE START
189: FFT Input -> 0x000000003000000020000000100000000
190: FFT Input -> 0x000000007000000060000000500000004
191: FFT Input -> 0x000000000b0000000b0000000900000008
192: FFT Input -> 0x00000000f0000000e00000000d0000000c
238: FFT Output -> 0x00000000bfffffff400000004000000038
239: FFT Output -> 0x00000000bfffffff500000000fffffff0
240: FFT Output -> 0x00000000bfffffff8fffffff8
241: FFT Output -> 0x00000000bfffffff000000000
242: CORE DONE
Key Takeaways

• CoRAMs are “portals” to memory and I/O
  – familiar SRAM interface
  – memory shapes configurable to match application

• Control Threads express application’s memory access pattern in portable manner
  – simple subset of C

• Application accelerator still under your control
Break
(15 minutes)
CoRAM: Beneath the Hood
What “Runs” CoRAM?

Architecture
Microarchitecture

FPGA

Control thread programs

Network-on-Chip
Memory Translation (TLBs)
Memory Interfaces and Caches

Control Logic
Cluster DMA
Fabric

Control Logic
Cluster DMA
Fabric

Core Logic
SRAM
What “Runs” CoRAM?

Control thread programs

FPGA

Network-on-Chip

Memory Translation (TLBs)

Memory Interfaces and Caches
What “Runs” CoRAM?

Performance and Functionality Is Decoupled
CORFLOW: The CoRAM Compiler

- Portable Application
- Performance Specification
- H/W Spec
- FPGA Simulator

- RTL
- Control Threads

- CoRAM Mapper + Rewriter
- CONNECT NoC Generator
- DMA Clusters
- Control FSMs

- Platform Libraries
- EDA Tools

2/11/2013
CoRAM and CONNECT Tutorial / FPGA'13
CoRAM: Beneath the Hood

DMA Cluster

Network-on-Chip
(generated by CONNECT)
CoRAM: Beneath the Hood

DMA Cluster

Mem

R

Mem

R

Mem

R

Mem

R

...
The CONNECT NoC Generation Framework

http://users.ece.cmu.edu/~mpapamic/connect

Michael K. Papamichael
papamix@cs.cmu.edu
FPGAs and Networks-on-Chip (NoCs)

- Rapid growth of FPGA capacity and features
  - Extended SoC and full-system prototyping
  - FPGA-based high-performance computing

Need for flexible NoCs to support communication

Map existing ASIC-oriented NoC designs on FPGAs?

- Inefficient use of FPGA resources
- ASIC-driven NoC architecture not optimal for FPGA
FPGAs and Networks-on-Chip (NoCs)

- **Rapid growth of FPGA capacity and features**
  - Extended SoC and full-system prototyping
  - FPGA-based high-performance computing

Need for flexible NoCs to support communication

**Map existing ASIC-oriented NoC designs on FPGAs?**

- Inefficient use of FPGA resources
- ASIC-driven NoC architecture not optimal for FPGA

**CONNECT** FPGA-tuned NoC Architecture

- Embodies FPGA-motivated design principles
- Very lightweight, minimizes resource usage
  - ~50% resource reduction vs. ASIC-oriented NoC
- Publicly released flexible NoC generator (demo)

Often goes against ASIC-driven NoC conventional wisdom
Outline

- **NoC Terminology** (single-slide review)
- **CONNECT** Approach
  - Tailoring the NoC to the FPGA Substrate
  - Application-specific NoC Tuning
- Public Release and Demo!
Outline

- **NoC Terminology** (single-slide review)

- **CONNECT** Approach
  - Tailoring the NoC to the FPGA Substrate
  - Application-specific NoC Tuning

- Related Work & Conclusion

- Public Release & Demo!
NoC Terminology Overview

- **Packets**
  - Basic logical unit of transmission

- **Flits**
  - Packets broken into into multiple *flits* – unit of flow control

- **Virtual Channels**
  - Multiple logical channels over single physical link

- **Flow Control**
  - Management of buffer space in the network
Outline

- NoC Terminology (single-slide review)

**CONNECT Approach**

- Tailoring the NoC to the FPGA Substrate
- Application-specific NoC Tuning

- Public Release and Demo!
How FPGAs are Different from ASICs

- FPGAs peculiar HW realization substrate in terms of:
  - Relative cost of speed vs. logic vs. wires vs. memory
  - Unique mapping and operating characteristics

**CONNECT** focuses on 4 FPGA characteristics:

1. Abundance of Wires
2. Storage Shortage & Peculiarities
3. Frequency Challenged
4. Reconfigurable Nature

FPGA characteristics uniquely influence key NoC design decisions
Tailoring NoCs to FPGAs

**Abundance of Wires**

- Densely connected wiring substrate
  - (Over)provisioned to handle worst case
  - Wires are “free” compared to other resources

**CONNECt NoC Implications**

- Make datapaths and channels as wide as possible
- Adjust packet format
  - E.g. carry control info on the side through dedicated links
- Adapt traditional credit-based flow control
Tailoring NoCs to FPGAs

Storage Shortage & Peculiarities

- Modern FPGAs offer storage in two forms
  - Block RAMs and LUT RAMs (use logic resources)
  - Only come in specific aspect ratios and sizes
- Typically in high demand, especially Block RAMs

CONNECT NoC Implications

- Minimize usage and optimize for aspect ratios and sizes
  - Implement multiple logical flit buffers in each physical buffer
- Use LUT RAM for flit buffers
  - Block RAM much larger than typical NoC flit buffer sizes
  - Allow rest of design to use scarce Block RAM resources
Tailoring NoCs to FPGAs

**Frequency “Challenged”**

- Much lower frequencies compared to ASICs
  - LUTs inherently slower than ASIC standard cells
  - Large wire delays when chaining LUTs
- Rapidly diminishing returns when pipelining
  - Deep pipelining hard due to quantization effects

**CONNECT NoC Implications**

- Default is single-stage pipeline. Pipeline as needed.
  - Also dramatically reduces network latency
- Make up for lower frequency by adjusting network
  - E.g. increase width of datapath and links or change topology
Tailoring NoCs to FPGAs

Reconfigurable Nature

- Reconfigurable nature of FPGAs
  - Sets them apart from ASICs
  - Support diverse range of applications

CONNECT NoC Implications

- Support extensive application-specific customization
  - Flexible parameterized NoC architecture
  - Automated NoC design generator (demo!)
- Adhere to standard common interface
  - NoC appears as plug-and-play black box from user-perspective
CONNECT vs. ASIC-Oriented RTL

16-node 4x4 Mesh Network-on-Chip (NoC)

- **SOTA:** state-of-the-art high-quality ASIC-oriented RTL*
- **CONNECT:** identically configured CONNECT-generated RTL

FPGA Resource Usage
(same router/NoC configuration)

*NoC RTL from http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router
CONNECT vs. ASIC-Oriented RTL

16-node 4x4 Mesh Network-on-Chip (NoC)

- **SOTA**: state-of-the-art high-quality ASIC-oriented RTL*
- **CONNECT**: identically configured CONNECT-generated RTL

FPGA Resource Usage
(same router/NoC configuration)

- ~50% lower LUT Usage

Network Performance
(uniform random traffic @ 100MHz)

- Similar bandwidth
- Good NoCs
- High saturation BW
- Lower latency
- Lower idle latency
- 4x BW within budget?

*NoC RTL from http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router
**CONNECT Sample Networks**

- Four sample CONNECT Networks (router, endpoint)
- 16 endpoints, 2/4 virtual channels, 128-bit datapath

![Ring Network](image1)
![Fat Tree Network](image2)
![Mesh Network](image3)
![High Radix Network](image4)

All above networks are interchangeable from user perspective.

<table>
<thead>
<tr>
<th>Load (in flits/cycle)</th>
<th>Latency (in cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>0.50</td>
</tr>
<tr>
<td>0.75</td>
<td>1.00</td>
</tr>
</tbody>
</table>

There is no one-size-fits-all NoC! Tune NoC to application.
Outline

- NoC Terminology (single-slide review)

**CONNECT** Approach

- Tailoring the NoC to the FPGA Substrate
- Application-specific NoC Tuning

Public Release and Demo!
Application-Specific NoC Tuning

- NoC typically viewed as black-box in FPGA applications
  - NoC provides connectivity - topology, router, other details hidden
  - Left-over mentality from vendor-supplied interconnect solutions
  - Compromises design and fine tuning for application generality

  Black-box approach inefficient and not scalable

- Need for application-specific NoC Tuning enabled by
  - Reconfigurable FPGA environment
  - Traffic pattern and connectivity/communication requirements
  - Application-level observations

CONNECT NoC tuning within CoRAM

- Tuning NoC at all levels, such as topology, router, flow-control

Demonstrates the effectiveness of hand-in-hand NoC tuning
Network Optimization Gains

- CoRAM application with 16 endpoints
  - 4 memory controllers and 12 clusters (6 read, 6 write)
  - Three network options, equivalent from CoRAM perspective

![Network Diagram]

<table>
<thead>
<tr>
<th>Network</th>
<th>Configuration</th>
<th>Xilinx Virtex-6 LX760T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Router: 4x4 Mesh</td>
<td>LUTs: 46361 Freq. (MHz): 112</td>
</tr>
<tr>
<td></td>
<td>Flow Control: Credits</td>
<td></td>
</tr>
<tr>
<td>Symmetric Tree</td>
<td>Router: VOQ-based</td>
<td>LUTs: 5137 Freq. (MHz): 126</td>
</tr>
<tr>
<td></td>
<td>Flow Control: Peek</td>
<td></td>
</tr>
<tr>
<td>Asymmetric Tree</td>
<td>Router: VOQ-based</td>
<td>LUTs: 3642 Freq. (MHz): 220</td>
</tr>
<tr>
<td></td>
<td>Flow Control: Peek</td>
<td></td>
</tr>
</tbody>
</table>

Order of magnitude reduction in FPGA resource usage

All networks assume 128-bit links and equal buffering per router
Asymmetric tree achieves same peak performance
Matrix Mult. Case Study – Area

Dense Matrix Multiply Area Comparison

FPGA LUTS

<table>
<thead>
<tr>
<th></th>
<th>Core Logic</th>
<th>Clusters</th>
<th>Network</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-D Mesh (8C)</td>
<td>150,000</td>
<td>50,000</td>
<td>50,000</td>
<td>50,000</td>
</tr>
<tr>
<td>Bidirectional Ring (8C)</td>
<td>100,000</td>
<td>25,000</td>
<td>25,000</td>
<td>25,000</td>
</tr>
<tr>
<td>Sym-Tree (8C)</td>
<td>120,000</td>
<td>20,000</td>
<td>20,000</td>
<td>20,000</td>
</tr>
<tr>
<td>Asym-Tree (4W-4R)</td>
<td>130,000</td>
<td>15,000</td>
<td>15,000</td>
<td>15,000</td>
</tr>
</tbody>
</table>

CoRAM FPGA overhead reduced by > 2x
Matrix Mult. Case Study – Efficiency

Asymmetric approach 37% more efficient
Takeaway Points

- Significant gains from tuning for FPGA & application
  - FPGAs and ASICs have different design “sweet spot”

- CONNECT → flexible, efficient, lightweight NoC

- Compared to ASIC-driven NoC, CONNECT offers
  - Significantly lower network latency and
  - ~50% lower LUT usage or 3-4x higher network performance

- Take advantage of reconfigurable nature of FPGA
  - Tailor NoC to specific communication needs of application
Outline

- NoC Terminology (single-slide review)

**CONNECT** Approach

- Tailoring the NoC to the FPGA Substrate
- Application-specific NoC Tuning

Public Release and Demo!
NoC Generator with web-based interface
- Supports multiple pre-configured topologies
- Includes graphical editor for custom topologies
- FreeBSD-like license (limited to non-commercial research use)

Acknowledgments
- Derek Chiou, Daniel Becker & Stanford CVA group
- NSF, Xilinx, Bluespec

Demo!
CONNECT

Some Release Stats

- Released in March 2012
  - 2500+ unique visitors
  - 300+ network generation requests

Most Popular Topologies

1. Mesh/Torus – 51%
2. Double Ring – 14%
3. Ring/Line – 14%
4. Fully Connected – 10%
5. Custom – 6%
6. Star – 5%

User Breakdown

- 35% Other
- 40% Academia
- 25% Industry
Advanced Topics
We Have Only Scratched the Surface

• Not enough time to cover everything
  – running on real hardware (I/O and memory loading)
  – memory personality libraries
  – cluster and NoC tuning
  – control thread synthesis and tuning
  – high-level synthesis (Bluespec, C-to-CoRAM)

• Last lesson for today
  – simple tuning using H/W spec file
  – compare area and performance
How To “Tune” CoRAM?

- CoRAM microarchitecture is fully controllable
  - hardware constraints defined by H/W spec file
    (e.g., max clusters, max membanks, NoC topology)

Low Memory Intensity
- Simple ring network
- Single memory bank
- Single DMA cluster
- Few CoRAMs

High Memory Intensity
- Aggressive topology (Mesh, Xbar)
- Non-blocking control threads
- More banks/DMA clusters
- Wider NoC Links
- Many CoRAMs
DFT-64 (dual-core)

• Look at new DFT-64 example:
  $ cd ~/coram/project/DftDemoDual
  $ emacs DftDemoDual.c &
  $ emacs DftDemoDual.v &

• What’s different from before:
  – Now have 2 DFT-64 cores
    (create more “traffic”)
  – 4 CoRAMs (2 per core)
  – 1 control thread with non-blocking control actions
The CoRAM H/W Spec File

- Open the H/W spec file and scroll to the bottom:
  
  ```
  $ emacs ~/coram/proj/DftDemoDual/DftDemoDual.spec 
  ```

- Parameters:
  - `num_mc=1`
  - `ports_per_mc=4`
  - `max_rdwr_cluster=1`
  - `topology=ring`

![Diagram of CoRAM and CONNECT Tutorial / FPGA'13](image-url)
Run and Measure

• Build and run the new DFT example
  
  $ make DftDemoDual-corflow
  $ cd DftDemoDual-corflow
  $ ./run_icarus.sh

• Expected Output:
  – [Ctrl-Thread] End cycle: 537
Let’s Try Different Approach

Change 2 lines below in DftDemoDual.spec

```plaintext
max_rdwr_cluster=2
topology=xbar
```

Repeat Earlier Steps to Run, Simulate
A Closer Look (Waveform)

1 cluster, ring:

2 clusters, crossbar:
How Do I Get CoRAM?

• We are accepting beta users
  – Register your name and affiliation with us
  – Send request to coram@ece.cmu.edu

• Required tools that we don’t provide:
  – Bluespec Compiler (2012.09beta)
  – Xilinx ISE 13.1 or higher
  – LLVM 2.8, GCC
Supported Platforms

• Stable targets supported by CoRAM
  – Xilinx ML605
  – Altera/Terasic DE4

• In progress
  – ZYNQ ZC702
  – ZedBoard
  – Convey HC-*
Summary

• The CoRAM Architecture
  – standard, scalable memory architecture for FPGA
  – simplifies application development and portability

• Work-in-Progress
  – app-specific NoC customization (Michael)
  – compiler-managed query accelerators (Eric)
  – auto-generation of optimized systems from C (Gabe)
DEMO