Lecture 13: (Re)configurable Computing

Prof. Jan Rabaey
Computer Science 252, Spring 2000

The major contributions of Andre Dehon to this slide set are gratefully acknowledged

Computers in the News …

TI announces 2 new DSPs

- **C64x**
  - Up to 1.1 GHz
  - 9 Billion Operations/sec
  - 10x performance of C62x
  - 32 full-rate DSL modems on a single chip!

- **C55x**
  - 0.05 mW/MIPS (20 MIPS/mW!)
  - Cut power consumption of C54x by 85%
  - 5x performance of C54x
C64x

Enhanced performance for communications and multimedia

Communications
750MHz C64x vs. 300 MHz C62x™

- 12x
- 5x
- 5x
- 5x

Filter Reed Solomon

Viterbi

FFT

IDCT

Morphology

Imaging/Video
750MHz C64x vs. 300 MHz C62x™

- 19x
- 15x
- 5x
- 5x

Motion Estimation
From the C54x core …

To the C55x
Leading to higher energy efficiency (?)

Evaluation metrics for Embedded Systems

- Components of Cost
  - Area of die / yield
  - Code density (memory is the major part of die size)
  - Packaging
  - Design effort
  - Programming cost
  - Time-to-market
  - Reusability

Performance as a Functionality Constraint
(“Just-in-Time Computing”)
**Special Instructions for Specific Applications**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMPY4</td>
<td>Galois Field MPY</td>
<td>Reed Solomon Support</td>
</tr>
<tr>
<td>SUBABS4</td>
<td>Quad 8-bit Absolute of Differences</td>
<td>Motion Estimation</td>
</tr>
<tr>
<td>SSHVL, SSHVR</td>
<td>Signed Variable Shift</td>
<td></td>
</tr>
<tr>
<td>SHFL</td>
<td>Bit Interleaving</td>
<td></td>
</tr>
<tr>
<td>DEAL</td>
<td>Bit De-Interleaving</td>
<td></td>
</tr>
<tr>
<td>SWAP4</td>
<td>Byte Swap</td>
<td></td>
</tr>
<tr>
<td>XPNDx</td>
<td>Bit Expansion</td>
<td></td>
</tr>
<tr>
<td>MPYHx</td>
<td>Extended Precision</td>
<td></td>
</tr>
<tr>
<td>MPYULx</td>
<td>16x52 MPYys</td>
<td></td>
</tr>
<tr>
<td>AVGx</td>
<td>Quad 9-bit, Dual 16-bit Average</td>
<td></td>
</tr>
<tr>
<td>BTC4</td>
<td>Bit Count</td>
<td>Machine Vision</td>
</tr>
</tbody>
</table>

**What is Configurable Computing?**

Spatially-programmed connection of processing elements

"Hardware" customized to specifics of problem. Direct map of problem specific dataflow, control. Circuits “adapted” as problem requirements change.
**Spatial vs. Temporal Computing**

**Spatial**
- Computes one function (e.g., FP-multiply, divider, DCT)
- Function defined at fabrication time

**Temporal**
- Computes “any” computable function (e.g., Processor, DSPs, FPGAs)
- Function defined after fabrication

---

**Defining Terms**

**Fixed Function:**
- Computes one function (e.g., FP-multiply, divider, DCT)
- Function defined at fabrication time

**Programmable:**
- Computes “any” computable function (e.g., Processor, DSPs, FPGAs)
- Function defined after fabrication

Parameterizable Hardware:
Performs limited “set” of functions
“Any” Computation?  
(Universality)

• Any computation which can “fit” on the programmable substrate  
• Limitations: hold entire computation and intermediate data  
• Recall size/fit constraint

Benefits of Programmable

• Non-permanent customization and application development after fabrication  
  - “Late Binding”  
• economies of scale (amortize large, fixed design costs)  
• time-to-market (evolving requirements and standards, new ideas)

Disadvantages

• Efficiency penalty (area, performance, power)  
• Correctness Verification
Spatial/Configurable Benefits

- 10x raw density advantage over processors
- Potential for fine-grained (bit-level) control --- can offer another order of magnitude benefit
- Locality!

Spatial/Configurable Drawbacks

- Each compute/interconnect resource dedicated to single function
- Must dedicate resources for every computational subtask
- Infrequently needed portions of a computation sit idle --> inefficient use of resources

Density Comparison

- Graph showing computational density vs technology for SRAM-based FPGAs and RISC Processors.
### Processor vs. FPGA Area

![Diagram showing Processor vs. FPGA Area](image)

### Processors and FPGAs

<table>
<thead>
<tr>
<th>Year</th>
<th>Design</th>
<th>Organization</th>
<th>( \lambda )</th>
<th>( \lambda^2 ) area</th>
<th>cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1984</td>
<td>MIPS</td>
<td>1 x 32</td>
<td>1.5M</td>
<td>15M</td>
<td>250ns</td>
</tr>
<tr>
<td>1987</td>
<td>MIPS-X</td>
<td>1 x 32</td>
<td>1.0M</td>
<td>68M</td>
<td>50ns</td>
</tr>
<tr>
<td>1994</td>
<td>MIPS</td>
<td>1 x 32</td>
<td>0.28M</td>
<td>1.7G</td>
<td>2ns</td>
</tr>
<tr>
<td>1992</td>
<td>Alpha</td>
<td>1 x 64</td>
<td>0.38M</td>
<td>1.7G</td>
<td>5ns</td>
</tr>
<tr>
<td>1995</td>
<td>Alpha</td>
<td>2 x 64</td>
<td>0.25M</td>
<td>4.8G</td>
<td>3.3ns</td>
</tr>
<tr>
<td>1996</td>
<td>Alpha</td>
<td>2 x 64</td>
<td>0.18M</td>
<td>6.8G</td>
<td>2.3ns</td>
</tr>
<tr>
<td>1992</td>
<td>PADDI</td>
<td>8 x 16</td>
<td>0.6M</td>
<td>126M</td>
<td>40ns</td>
</tr>
<tr>
<td>1995</td>
<td>PADDI-2</td>
<td>48 x 16</td>
<td>0.5M</td>
<td>515M</td>
<td>20ns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Design</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1986</td>
<td>Xilinx</td>
<td>2K 1 CLB (4 LUT)</td>
</tr>
<tr>
<td>1988</td>
<td>Xilinx</td>
<td>3K 64 CLBs (2.4-LUT)</td>
</tr>
<tr>
<td>1992</td>
<td>Xilinx</td>
<td>4K 49 CLBs (2.4-LUT)</td>
</tr>
<tr>
<td>1995</td>
<td>Xilinx</td>
<td>5K 49 CLBs (4.4-LUT)</td>
</tr>
</tbody>
</table>
**Early RC Successes**

- Fastest RSA implementation is on a reconfigurable machine (DEC PAM)
- Splash2 (SRC) performs DNA Sequence matching 300x Cray2 speed, and 200x a 16K CM2
- Many modern processors and ASICs are verified using FPGA emulation systems
- For many signal processing/filtering operations, single chip FPGAs outperform DSPs by 10-100x.

**Issues in Configurable Design**

- Choice and Granularity of Computational Elements
- Choice and Granularity of Interconnect Network
- (Re)configuration Time and Rate
  - Fabrication time --> Fixed function devices
  - Beginning of product use --> Actel/Quicklogic FPGAs
  - Beginning of usage epoch --> (Re)configurable FPGAs
  - Every cycle --> traditional Instruction Set Processors
The Choice of the Computational Elements

FPGA Basics

- LUT for compute
- FF for timing/retiming
- Switchable interconnect
- ...everything we need to build fixed logic circuits
  - don’t really need programmable gates
  - latches can be built from gates
Field Programmable Gate Array (FPGA) Basics

Collection of programmable “gates” embedded in a flexible interconnect network.
...a “user programmable” alternative to gate arrays.

Look-Up Table (LUT)

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

2-LUT
**LUTs**

- K-LUT -- K input lookup table
- Any function of K inputs by programming table

**Conventional FPGA Tile**

K-LUT (typical k=4)

w/ optional output Flip-Flop
Commercial FPGA (XC4K)

- Cascaded 4 LUTs (2 4-LUTs -> 1 3-LUT)
- Fast Carry support
- Segmented interconnect
- Can use LUT config as memory.
**Not Restricted to Logic Gates**

*Example: Paddi-2 (1995)*

![Diagram of a Communication Network](image)

**A Data-driven Computation Paradigm**

![Diagram of a data-driven computation](image)
Not restricted to Logic Gate Operations

For Spatial Architectures

• Interconnect dominant
  – area
  – power
  – time

• …so need to understand in order to optimize architectures
**Dominant in Area**

\[ A_{\text{dominant}} = A_{f_{\text{fixed}}} + \frac{N_{\text{SHF}}(N_p, w, p) \cdot A_{\text{SHF}}}{\text{interconnect}} + \left( \frac{C}{M} \right) \cdot n_{\text{bus}} \cdot A_{\text{mem-cell}} + d \cdot A_{\text{mem-cell}} + \text{re timing memory} \]

<table>
<thead>
<tr>
<th>Function</th>
<th>Area ($\mu^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUT MUX + ff</td>
<td>20K (generous, closer to 10K)</td>
</tr>
<tr>
<td>Programming Memory</td>
<td>80K (240K typical unencoded)</td>
</tr>
<tr>
<td>Interconnect</td>
<td>700K (for $N_p = 2048$)</td>
</tr>
</tbody>
</table>

**Dominant in Time**

<table>
<thead>
<tr>
<th>Design</th>
<th>Path</th>
<th>Total Delay</th>
<th>LUT Delay</th>
<th>Inter. %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera 10K130V-2</td>
<td>LUT-local-LUT</td>
<td>2.5 ns</td>
<td>2.1 ns</td>
<td>16%</td>
</tr>
<tr>
<td></td>
<td>LUT-row-local-LUT</td>
<td>6.6 ns</td>
<td>2.1 ns</td>
<td>68%</td>
</tr>
<tr>
<td></td>
<td>LUT-column-local-LUT</td>
<td>11.1 ns</td>
<td>2.1 ns</td>
<td>81%</td>
</tr>
<tr>
<td></td>
<td>LUT-row-column-local-LUT</td>
<td>15.6 ns</td>
<td>2.1 ns</td>
<td>87%</td>
</tr>
<tr>
<td></td>
<td>LUT-row-fanout-local-LUT (fanout)</td>
<td>28 ns</td>
<td>2.1 ns</td>
<td>90%</td>
</tr>
</tbody>
</table>
Dominant in Power

<table>
<thead>
<tr>
<th>Interconnect</th>
<th>Clock</th>
<th>IO</th>
<th>CLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>65%</td>
<td>21%</td>
<td>9%</td>
<td>5%</td>
</tr>
</tbody>
</table>

XC4003A data from Eric Kusse (UCB MS 1997)

Interconnect

- Problem
  - Thousands of independent (bit) operators producing results
    - true of FPGAs today
    - ...true for LIW, multi-uP, etc. in future
  - Each taking as inputs the results of other (bit) processing elements
  - Interconnect is late bound
    - don’t know until after fabrication
**Design Issues**

- Flexibility -- route "anything"
  - (w/in reason?)
- Area -- wires, switches
- Delay -- switches in path, stubs, wire length
- Power -- switch, wire capacitance
- Routability -- computational difficulty finding routes

**First Attempt: Crossbar**

- Any operator may consume output from any other operator
- Try a crossbar?
Crossbar

- Flexibility (++)
  - routes everything (guaranteed)
- Delay (Power) (-)
  - wire length O(kn)
  - parasitic stubs: kn+n
  - series switch: 1
  - O(kn)

- Area (-)
  - Bisection bandwidth n
  - kn² switches
  - O(n²)

Too expensive and not scalable

Avoiding Crossbar Costs

- Good architectural design
  - Optimize for the common case
- Designs have spatial locality
- We have freedom in operator placement
- Thus: Place connected components “close” together
  - don’t need full interconnect?
Exploit Locality

- Wires expensive
- Local interconnect cheap
- Try a mesh?

The Toronto Model

Switch Box
Connect Box
Logic Tile

JR.S00 41
JR.S00 42
Mesh Analysis

• Flexibility - ?
  - Ok w/ large w
• Delay (Power)
  - Series switches
    - 1/√n
  - Wire length
    - w/√n
  - Stubs
    - O(w)–O(w√n)

  - Area
    - Bisection BW = w√n
    - Switches = O(nw)
    - O(w^2n)

• Can we place everything close?
**Mesh “Closeness”**

- Try placing “everything” close

<table>
<thead>
<tr>
<th>Manhattan Distance</th>
<th>Transitive Places</th>
<th>Fanin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>64</td>
</tr>
<tr>
<td>( n )</td>
<td>( 4^n )</td>
<td>( 4^n )</td>
</tr>
</tbody>
</table>

**Adding Nearest Neighbor Connections**

- Connection to 8 neighbors
- Improvement over Mesh by x3
- Good for neighbor-neighbor connections
Typical Extensions

- Segmented Interconnect
- Hardwired/Cascade Inputs

XC4K Interconnect
Creating Hierarchy

Example: Paddi-2

Level-2 Network
16 x 16b

Level-1 Network
6 x 16b

16 x 6 switch matrix
**Level-1 Communication Network**

- 1-cycle Latency
- Full Connectivity
- On top of Data Path in Metal-2

**Level-2 Communication Network (Pipelined)**

- 8 x 16b data buses
- 8 x 1b ctrl buses
- 8 x 16b data buses
- Programmable switches
**Paddi-2 Processor**

- 1-µm 2-metal CMOS tech
- 1.2 x 1.2 mm²
- 600k transistors
- 208-pin PGA
- \(f_{\text{clock}} = 50\) MHz
- \(P_{\text{av}} = 3.6 \text{ W} @ 5V\)

**How to Provide Scalability?**

- Tree of Meshes

Main question:
How to populate/parameterize the tree?
Hierarchical Interconnect

- Two regions of connectivity lengths
- Hybrid architecture using both Mesh and Binary structures favored

Hybrid Architecture Revisited

Straightforward combination of Mesh and Binary tree is not smart

- Short connections will be through the Mesh architecture
- The cheap connections on the Binary tree will be redundant
Inverse Clustering

- Blocks further away are connected at the lowest levels
- Inverse clustering complements Mesh Architecture

Hybrid Interconnect Architecture

- Levels of interconnect targeting different connectivity lengths

<table>
<thead>
<tr>
<th>Level0</th>
<th>Level1</th>
<th>Level2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest Neighbor</td>
<td>Mesh Interconnect</td>
<td>Hierarchical</td>
</tr>
</tbody>
</table>

Manhattan Distance

Energy x Delay

Mesh

Binary Tree

Mesh + Inverse

JR.S00 57

JR.S00 58
Prototype

- Array Size: 8x8 (2 x 4 LUT)
- Power Supply: 1.5V & 0.8V
- Configuration: Mapped as RAM
- Toggle Frequency: 125MHz
- Area: 3mm x 3mm
- Process: 0.25U ST

Programming the Configurable Platform

- RTL
  - Tech. Indep. Optimization → LUT Mapping
  - Placement → Routing
  - Bitstream Generation → Config. Data
**Starting Point**

- RTL
  - $t = A + B$
  - $Reg(t, C, clk)$
- Logic
  - $O_{i+1} = A_i \oplus B_i \oplus C_i$
  - $C_{i+1} = A_i B_i \lor B_i C_i \lor A_i C_i$

**LUT Map**
Placement

- Maximize locality
  - minimize number of wires in each channel
  - minimize length of wires
  - (but, cannot put everything close)
- Often start by partitioning/clustering
- State-of-the-art finish via simulated annealing
Routing

- Often done in two passes
  - Global to determine channel
  - Detailed to determine actual wires and switches

- Difficulty is
  - limited channels
  - switchbox connectivity restrictions
Summary

- Configurable Computing using “programming in space” versus “programming in time” for traditional instruction-set computers
- Key design choices
  - Computational units and their granularity
  - Interconnect Network
  - (Re)configuration time and frequency
- Next class: Some practical examples of reconfigurable computers