Quick Points

- HW #3, #4 graded and returned
- Next week Thursday, project status updates
  - 10 minute presentations per group + questions
  - Upload to WebCT by the previous evening
- Expected that you’ve made some progress!

Allowable Schedules

Active LUTs ($N_A$) = 3

Sequentialization

- Adding time slots
  - More sequential (more latency)
  - Adds slack
    - Allows better balance

Multicontext Scheduling

- “Retiming” for multicontext
  - **goal**: minimize peak resource requirements
  - NP-complete
  - List schedule, anneal
    - How do we accommodate intermediate data?
    - Effects?

Signal Retiming

- Non-pipelined
  - hold value on LUT Output (wire)
    - from production through consumption
  - Wastes wire and switches by occupying
    - For entire critical path delay L
    - Not just for 1/L'th of cycle takes to cross wire segment
  - How will it show up in multicontext?
Signal Retiming

- Multicontext equivalent
  - Need LUT to hold value for each intermediate context

Multicontext Version

- Three contexts:
  12 LUTs @ 1040K\(\lambda^2\) = 12.5M\(\lambda^2\)
- Pipelining needed for dependent paths

ASCII→Hex Example

- All retiming on wires (active outputs)
  - Saturation based on inputs to largest stage
  - With enough contexts only one LUT needed
  - Increased LUT area due to additional stored configuration information
  - Eventually additional interconnect savings taken up by LUT configuration overhead

General Throughput Mapping

- If only want to achieve limited throughput
- Target produce new result every \(t\) cycles
- Spatially pipeline every \(t\) stages
  - \(\text{cycle} = t\)
- Retime to minimize register requirements
- Multicontext evaluation w/in a spatial stage
  - Retime (list schedule) to minimize resource usage
- Map for depth (i) and contexts (c)
Benchmark Set

- 23 MCNC circuits
  - Area mapped with SIS and Chortle

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Mapped UFs</th>
<th>Path Length</th>
<th>Circuit</th>
<th>Mapped UFs</th>
<th>Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>89ifm</td>
<td>46</td>
<td>10</td>
<td>92ifac</td>
<td>1267</td>
<td>13</td>
</tr>
<tr>
<td>96mm</td>
<td>123</td>
<td>7</td>
<td>94ifm</td>
<td>230</td>
<td>9</td>
</tr>
<tr>
<td>99interm</td>
<td>108</td>
<td>8</td>
<td>95ifm</td>
<td>45</td>
<td>17</td>
</tr>
<tr>
<td>C990</td>
<td>85</td>
<td>10</td>
<td>minmix</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>C990</td>
<td>175</td>
<td>21</td>
<td>mistex2</td>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td>99ifm</td>
<td>169</td>
<td>19</td>
<td>sif</td>
<td>105</td>
<td>10</td>
</tr>
<tr>
<td>apex6</td>
<td>148</td>
<td>9</td>
<td>sif</td>
<td>150</td>
<td>9</td>
</tr>
<tr>
<td>apex7</td>
<td>77</td>
<td>7</td>
<td>sof</td>
<td>203</td>
<td>16</td>
</tr>
<tr>
<td>bf9</td>
<td>46</td>
<td>7</td>
<td>sif</td>
<td>73</td>
<td>9</td>
</tr>
<tr>
<td>c99</td>
<td>121</td>
<td>9</td>
<td>vgf</td>
<td>60</td>
<td>9</td>
</tr>
<tr>
<td>condic</td>
<td>367</td>
<td>13</td>
<td>jen</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>count1</td>
<td>46</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Area v. Throughput

Area v. Throughput (cont.)

Reconfiguration for Fault Tolerance

- Embedded systems require high reliability in the presence of transient or permanent faults
- FPGAs contain substantial redundancy
- Possible to dynamically "configure around" problem areas
- Numerous on-line and off-line solutions

Column Based Reconfiguration

- Huang and McCluskey
- Assume that each FPGA column is equivalent in terms of logic and routing
  - Preserve empty columns for future use
  - Somewhat wasteful
- Precompile and compress differences in bitstreams

Column Based Reconfiguration

- Create multiple copies of the same design with different unused columns
- Only requires different inter-block connections
- Can lead to unreasonable configuration count
Column Based Reconfiguration

- Determining differences and compressing the results leads to "reasonable" overhead
- Scalability and fault diagnosis are issues

![Column Based Reconfiguration Diagram](image)

Summary

- In many cases cannot profitably reuse logic at device cycle rate
  - Cycles, no data parallelism
  - Low throughput, unstructured
  - Dissimilar data dependent computations
- These cases benefit from having more than one instructions/operations per active element
- Economical retiming becomes important here to achieve active LUT reduction
- For \( c=[4,8] \), \( l=[4,6] \) automatically mapped designs are 1/2 to 1/3 single context size

Outline

- Continuation
  - Function Unit Architectures
    - Motivation
    - Various architectures
    - Device trends

Coarse-grained Architectures

- DP-FPGA
  - LUT-based
  - LUTs share configuration bits
- Rapid
  - Specialized ALUs, multipliers
  - 1D pipeline
  - Matrix
    - 2-D array of ALUs
  - Chess
    - Augmented, pipelined matrix
  - Raw
    - Full RISC core as basic block
    - Static scheduling used for communication

Configuration Sharing

- Set \( MC = 2 \times CE \)
Two-dimensional Layout

- Control network supports distributed signals
- Data routed as four-bit values

DP-FPGA Technology Mapping

- Ideal case would be if all datapath divisible by 4, no "irregularities"
- Area improvement includes logic values only
- Shift logic included

RaPiD

- Reconfigurable Pipeline Datapath
- Ebeling – University of Washington
- Uses hard-coded functional units (ALU, Memory, multiply)
- Good for signal processing
- Linear array of processing elements

RaPiD Datapath

- Segmented linear architecture
- All RAMs and ALUs are pipelined
- Bus connectors also contain registers

RaPiD Control Path

- In addition to static control, control pipeline allows dynamic control
- LUTs provide simple programmability
- Cells can be chained together to form continuous pipe

FIR Filter Example

- Measure system response to input impulse
- Coefficients used to scale input
- Running sum determined total
FIR Filter Example (cont.)

Chain multiple taps together (one multiplier per tap)

MATRIX

- Dehon and Mirsky -> MIT
- 2-dimensional array of ALUs
- Each Basic Functional Unit contains “processor” (ALU + SRAM)
- Ideal for systolic and VLIW computation
- 8-bit computation
- Forerunner of SiliconSpice product

Basic Functional Unit

- Two inputs from adjacent blocks
- Local memory for instructions, data

MATRIX Interconnect

- Near-neighbor and quad connectivity
- Pipelined interconnect at ALU inputs
- Data transferred in 8-bit groups
- Interconnect not pipelined

Functional Unit Inputs

- Each ALU inputs come from several sources
- Note that source is locally configurable based on data values

FIR Filter Example

- For k-weight filter 4K cells needed
- One result every 2 cycles
- K/2 8x8 multiplies per cycle
Chess

- HP Labs – Bristol, England
- 2-D array – similar to Matrix
- Contains more “FPGA-like” routing resources
- No reported software or application results
- Doesn’t support incremental compilation

Chess Interconnect

- More like an FPGA
- Takes advantage of near-neighbor connectivity

Chess Basic Block

- Switchbox memory can be used as storage
- ALU core for computation

Reconfigurable Architecture Workstation

- MIT Computer Architecture Group
- Full RISC processor located as processing element
- Routing decoupled into switch mode
- Parallelizing compiler used to distribute work load
- Large amount of memory per tile

RAW Tile

- Full functionality in each tile
- Static router located for near-neighbor communication

RAW Datapath
Raw Compiler
- Parallelizes compilation across multiple tiles
- Orchestrates communication between tiles
- Some dynamic (data dependent) routing possible

Summary
- Architectures moving in the direction of coarse-grained blocks
- Latest trend is functional pipeline
- Communication determined at compile time
- Software support still a major issue