Previously

• Saw how to pipeline architectures
  – specifically interconnect
  – talked about general case
• Including how to map to them
• Saw how to reuse resources at maximum rate to do the same thing

Today

• Multicontext
  – Review why
  – Cost
  – Packing into contexts
  – Retiming requirements
• [concepts we saw in overview week 2-3, we can now dig deeper into details]

How often is reuse of the same operation applicable?

• In what cases can we exploit high-frequency, heavily pipelined operation?
  
  • …and when can we not?

How often is reuse of the same operation applicable?

• Can we exploit higher frequency offered?
  – High throughput, feed-forward (acyclic)
  – Cycles in flowgraph
    • abundant data level parallelism [C-slow]
    • no data level parallelism
  – Low throughput tasks
    • structured (e.g. datapaths) [serialize datapath]
    • unstructured
  – Data dependent operations
    • similar ops [local control -- next time]
    • dissimilar ops

Structured Datapaths

• Datapaths: same pinst for all bits
• Can serialize and reuse the same data elements in succeeding cycles
  
  • example: adder
FPGA Model -- if throughput requirement is reduced for wide word operations, serialization allows us to reuse active area for same computation

Same graph, rotated to show backside.

Remaining Cases

• Benefit from multicontext as well as high clock rate
  • *i.e.*
    – cycles, no parallelism
    – data dependent, dissimilar operations
    – low throughput, irregular (can’t afford swap?)

Single Context

• When have:
  – cycles and no data parallelism
  – low throughput, unstructured tasks
  – dis-similar data dependent tasks
• Active resources sit idle most of the time
  – Waste of resources
• Cannot reuse resources to perform different function, only same

Resource Reuse

• To use resources in these cases
  – must direct to do different things.

• Must be able tell resources how to behave

  • separate instructions (*pinsts*) for each behavior

Example: Serial Evaluation
Example: Dis-similar Operations

Example: Multicontext Organization/Area
- $A_{\text{ctx}} = 80K_2$
- dense encoding
- $A_{\text{ctx}} : A_{\text{base}} = 1:10$
- $A_{\text{base}} = 800K_2$

Example: Multicontext Tradeoff Curves
- Assume Ideal packing: $N_{\text{active}} = N_{\text{total}} / L$
- In Practice
  - Limitations from:
    - Scheduling
    - Retiming

Reminder: Robust point: $c^* A_{\text{ctx}} = A_{\text{base}}$
Scheduling Limitations

- $N_A$ (active)
  - size of largest stage

- Precedence:
  - can evaluate a LUT only after predecessors have been evaluated
  - cannot always, completely equalize stage requirements

Scheduling

- Precedence limits packing freedom
- Freedom do have
  - shows up as slack in network

Computing Slack:

- ASAP (As Soon As Possible) Schedule
  - propagate depth forward from primary inputs
    - depth = 1 + max input depth
- ALAP (As Late As Possible) Schedule
  - propagate distance from outputs back from outputs
    - level = 1 + max output consumption level
- Slack
  - slack = $L + 1 - (\text{depth} + \text{level})$ [PI depth=0, PO level=0]

Allowable Schedules

Active LUTs ($N_A$) = 3

Sequentialization

- Adding time slots
  - more sequential (more latency)
  - add slack
    - allows better balance

$L = 4 \rightarrow N_A = 2$ (4 or 3 contexts)
Multicontext Scheduling

- "Retiming" for multicontext
  - goal: minimize peak resource requirements
    - resources: logic blocks, retiming inputs, interconnect

- NP-complete

- list schedule, anneal

Multicontext Data Retiming

- 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 show up in multicontext?

Signal Retiming

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

DES Latency Example

Alternate Retiming

- Recall from last time (Day 20)
  - Net buffer
    - smaller than LUT
  - Output retiming
    - may have to route multiple times
  - Input buffer chain
    - only need LUT every depth cycles
**Input Buffer Retiming**

- Can only take $K$ unique inputs per cycle
- Configuration depth differ from context-to-context

---

**ASCII→Hex Example**

*Single Context: 21 LUTs @ 880K$\lambda^2$=18.5M$\lambda^2$*

---

**ASCII→Hex Example**

*Three Contexts: 12 LUTs @ 1040K$\lambda^2$=12.5M$\lambda^2$*

---

**ASCII→Hex Example**

*All retiming on wires (active outputs)*
- saturation based on inputs to largest stage

---

**General throughput mapping:**

- If only want to achieve limited throughput
- Target produce new result every $t$ cycles
  1. Spatially pipeline every $t$ stages
cycle = $t$
  2. retime to minimize register requirements
  3. multicontext evaluation w/in a spatial stage
     retime (list schedule) to minimize resource usage
  4. 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 LUTs</th>
<th>Path Length</th>
<th>Circuit</th>
<th>Mapped LUTs</th>
<th>Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>70m</td>
<td>123</td>
<td>7</td>
<td>84m</td>
<td>108</td>
<td>8</td>
</tr>
<tr>
<td>84m</td>
<td>123</td>
<td>7</td>
<td>islm</td>
<td>1267</td>
<td>13</td>
</tr>
<tr>
<td>C446</td>
<td>85</td>
<td>10</td>
<td>miscex1</td>
<td>23</td>
<td>6</td>
</tr>
<tr>
<td>C880</td>
<td>175</td>
<td>21</td>
<td>miscex2</td>
<td>38</td>
<td>8</td>
</tr>
<tr>
<td>cut2</td>
<td>169</td>
<td>19</td>
<td>stdf3</td>
<td>105</td>
<td>10</td>
</tr>
<tr>
<td>pscvls</td>
<td>248</td>
<td>9</td>
<td>stdf4</td>
<td>150</td>
<td>9</td>
</tr>
<tr>
<td>pspvls</td>
<td>77</td>
<td>7</td>
<td>stdf7</td>
<td>203</td>
<td>16</td>
</tr>
<tr>
<td>braf</td>
<td>46</td>
<td>7</td>
<td>sasq2</td>
<td>73</td>
<td>9</td>
</tr>
<tr>
<td>clipb</td>
<td>121</td>
<td>9</td>
<td>wcq2</td>
<td>60</td>
<td>9</td>
</tr>
<tr>
<td>condic</td>
<td>367</td>
<td>13</td>
<td>z4mil</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

Multicontext vs. Throughput

General Theme

- Ideal Benefit
  - e.g. Active=N/C
- Precedence Constraints
- Resource Limits
  - Sometimes bottleneck
- Net Benefit
- Resource Balance

Big Ideas

[MSB Ideas]

- Several cases cannot profitably reuse same logic at device cycle rate
  - cycles, no data parallelism
  - low throughput, unstructured
  - dis-similar data dependent computations
- These cases benefit from more than one instructions/operations per active element
  - $A_{\text{obt}} << A_{\text{active}}$ makes interesting
  - save area by sharing active among instructions

Admin

- Assignment 8 due Wednesday
- Still swapping around lectures
  - Note reading for Wed. online
- Final Exercise out Wednesday
  - …on everything…
  - …but includes time multiplexing
Big Ideas
[MSB-1 Ideas]

- Economical retiming becomes important here to achieve active LUT reduction
  - one output reg/LUT leads to early saturation
- c=4--8, l=4--6 automatically mapped designs 1/2 to 1/3 single context size
- Most FPGAs typically run in realm where multicontext is smaller
  - How many for intrinsic reasons?
  - How many for lack of HSRA-like register/CAD support?