Previously

- Looked broadly at instruction effects
- Looked at structural components of computation
  - interconnect
  - compute
  - retiming
- Looked at time-multiplexing

Today

- Control
  - data-dependent operations
- Different forms
  - local
  - instruction selection
- Architectural Issues

Control

- **Viewpoint**: can have instruction stream sequence without control
  - *i.e.* static/data-independent progression through sequence of instructions is control free
    - C0→C1→C2→C0→C1→C2→C0→...
    - Similarly, FSM w/ no data inputs
    - *E.g.* Day 5...non-branching datapath

Our “First” Programmable Architecture

Day 5
Terminology (reminder)

- **Primitive Instruction** \((\text{pinst})\)
  - Collection of bits which tell a bit-processing element what to do
  - Includes:
    - select compute operation
    - input sources in space (interconnect)
    - input sources in time (retiming)

- **Configuration Context**
  - Collection of all bits \((\text{pinsts})\) which describe machine’s behavior on one cycle

Why?

- Why do we need / want control?
- Static interconnect sufficient?
- Static sequencing?
- Static datapath operations?

Back to “Any” Computation

- Design must handle all potential inputs (computing scenarios)
- Requires sufficient generality
- However, computation for any given input may be much smaller than general case.
  - **Instantaneous** computation \(<\ll\) potential computation

Screwdriver Analogy

- Need capability to handle
  - Slothead
  - Phillips
  - Torq
  - Hex…
- But only need one at a time…

Video Decoder

- *E.g.* Video decoder \([\text{frame rate} = 33\text{ms}]\)
  - if (packet==FRAME)
    - if (type==I-FRAME)
      - I-FRAME computation
    - else if (type==B-FRAME)
      - B-FRAME computation
  - else if (type==B-FRAME)

Packet Processing

- If IP-V6 packet
  - ...
- If IP-V4 packet
  - ...
- If VoIP packet
  - ...
- If modem packet
  - ...
Two Control Options

1. Local control
   – unify choices
     • build all options into spatial compute structure and select operation
2. Instruction selection
   – provide a different instruction (instruction sequence) for each option
   – selection occurs when chose which instruction(s) to issue

FSM Example (local control)

FSM Example

Context 0 (S1=0)
Dout = 0
NS0 = /S0*Acyc*myAddr*Read
NS1 = S0

Context 1 (S1=1)
Dout = /S0
NS0 = /S0
NS1 = /S0

Local Control

• LUTs used ≠ LUT evaluations produced
  • → Counting LUTs not tell cycle-by-cycle LUT needs

FSM Example (Instruction)

Context 0 (S1=0)
Dout = 0
NS0 = /S0*Acyc*myAddr*Read
NS1 = S0

Context 1 (S1=1)
Dout = /S0
NS0 = /S0
NS1 = /S0

Local vs. Instruction

• If can decide early enough
  – and afford schedule/reload
  – instruction select → less computation
• If load too expensive
  – local instruction
    • faster
    • maybe even less capacity (AT)
How show up in modern μP?

• Instruction?

• Local control?

Slow Context Switch

• Instruction selection profitable only at coarse grain
  – Xilinx ms reconfiguration times
  – HSRA μs reconfiguration times
    • still 1000s of cycles
  • E.g. Video decoder [frame rate = 33ms]
    – if (packet==FRAME)
      • if (type==I-FRAME)
        – IF-context
      • else if (type==B-FRAME)
        – BF-context

Local vs. Instruction

• For multicontext device
  – i.e. fast (single) cycle switch
  – factor according to available contexts
• For conventional devices
  – factor only for gross differences
  – and early binding time

Optimization

• Basic Components
  – $T_{load}$ -- config. time
  – $T_{select}$ -- case compute time
  – $T_{gen}$ -- generalized compute time
  – $A_{select}$ -- case compute area
  – $A_{gen}$ -- generalized compute area

  • Minimize Capacity Consumed:
    – $AT_{local} = A_{gen} \times T_{gen}$
    – $AT_{select} =$
      • $A_{select}(T_{select}+T_{load})$
      • $T_{load}>0$ if can overlap w/ previous operation
        – know early enough
        – background load
        – have sufficient bandwidth to load

FSM Example

• FSM -- canonical “control” structure
  – captures many of these properties
  – can implement with deep multicontext
    • instruction selection
    – can implement as multilevel logic
      • unify, use local control

• Serve to build intuition
Full Partitioning Experiment

- Give each state its own context
- Optimize logic in state separately
- Tools
  - mustang, espresso, sis, Chortle
- Use:
  - one-hot encodings for single context
  - smallest/fastest
  - dense for multicontext
  - assume context select needs dense

Full Partitioning Experiment Results

<table>
<thead>
<tr>
<th>Name</th>
<th>Single Context</th>
<th>Context per State</th>
<th>Area</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.64</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Look at

- Assume stay in context for a number of LUT delays to evaluate logic/next state
- Pick delay from worst-case
- Assume single LUT-delay for context selection?
  - savings of 1 LUT-delay => comparable time
- Count LUTs in worst-case state

Full Partitioning

- Full partitioning comes out better
  - ~40% less area

- Note: full partition may not be optimal area case
  - e.g. intro example,
    - no reduction in area or time beyond 2-context implementation
  - 4-context (full partition) just more area
    - (additional contexts)

Partitioning versus Contexts (Area)

CSE benchmark

<table>
<thead>
<tr>
<th>Context</th>
<th>Area</th>
<th>Area [in kW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>1.6</td>
<td>0.8</td>
</tr>
<tr>
<td>6</td>
<td>2.8</td>
<td>1.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context</th>
<th>Area [in kW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>0.8</td>
</tr>
<tr>
<td>6</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Area vs. Context

- Area in kW

Area vs. Context

- Area in kW

Area vs. Context

- Area in kW

Area vs. Context

- Area in kW
Partitioning versus Contexts
(Delay)

CSE benchmark

Partitioning versus Contexts
(Heuristic)

- Start with dense mustang state encodings
- Greedily pick state bit which produces
  - least greatest area split
  - least greatest delay split
- Repeat until have desired number of contexts

Partition to Fixed Number of Contexts

<table>
<thead>
<tr>
<th>FSM</th>
<th>Best Single Context</th>
<th>Area Ratio by Number of Contexts</th>
<th>Dense Encodings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

N.B. - more realistic, device has fixed number of contexts.

Extend Comparison to Memory

- Fully local => compute with LUTs
- Fully partitioned => lookup logic (context) in memory and compute logic

- How compare to fully memory?
  - Simply lookup result in table?

Memory FSM Compare

<table>
<thead>
<tr>
<th>FSM</th>
<th>status</th>
<th>ns</th>
<th>outs</th>
<th>Mix area (M^2)</th>
<th>Integrated Adx &amp; Cota Organization</th>
<th>Memory area (M^2)</th>
<th>FPGA area (M^2)</th>
<th>8-ch FPGA area (M^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory FSM Compare (large)

<table>
<thead>
<tr>
<th>FSM</th>
<th>status</th>
<th>ns</th>
<th>outs</th>
<th>Mix area (M^2)</th>
<th>Integrated Adx &amp; Cota Organization</th>
<th>Memory area (M^2)</th>
<th>FPGA area (M^2)</th>
<th>8-ch FPGA area (M^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Memory FSM Compare (notes)
- Memory selected was “optimally” sized to problem
  - in practice, not get to pick memory allocation/organization for each FSM
  - no interconnect charged
- Memory operate in single cycle
  - but cycle slowing with inputs
- Smaller for <11 state+input bits
- Memory size not affected by CAD quality (FPGA/DPGA is)

Control Granularity
- What if we want to run multiple of these FSMs on the same component?
  - Local
  - Instruction

Local Control Multi-FSM
- Not rely on instructions
- Each wired up independently
- Easy to have multiple FSMs
  - (units of control)

Instruction Control
- If FSMs advance orthogonally
  - (really independent control)
  - context depth => product of states
    - for full partition
  - i.e. w/ single controller (PC)
    - must create product FSM
    - which may lead to state explosion
      - N FSMs, with S states => $S^N$ product states
    - This example:
      - 4 states, 2 FSMs => 16 state composite FSM
Architectural Questions

- How many pins/controller?
- Fixed or Configurable assignment of controllers to pins?
  - ...what level of granularity?

Architectural Questions

- Effects of:
  - Too many controllers?
  - Too few controllers?
  - Fixed controller assignment?
  - Configurable controller assignment?

Architectural Questions

- Too many:
  - wasted space on extra controllers
  - synchronization?
- Too few:
  - product state space and/or underuse logic
- Fixed:
  - underuse logic if when region too big
- Configurable:
  - cost interconnect, slower distribution

Control and FPGAs

- Local/single instruction not rely on controller
- Potential strength of FPGA
- Easy to breakup capacity and deploy to orthogonal tasks
- How processor handle orthogonal tasks?
  - Efficiency?

Control and FPGAs

- Data dependent selection
  - potentially fast w/ local control compared to \( \mu P \)
  - Can examine many bits and perform multi-way branch (output generation) in just a few LUT cycles
  - \( \mu P \) requires sequence of operations

Architecture Instr. Taxonomy

Control Threads (PCs)

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Architecture/Examples</th>
<th>Instruction Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 2</td>
<td>Hardwired Functional Unit (e.g. ECC/DEC Unit, FP/MPI)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>FPGA</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Reconfigurable ALUs</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>RhoSwift SVP</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Traditional Processors</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Vector Processors</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>GPPA</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TAP</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>VLV</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>RV64G8CORE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X16/V8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>VEGA</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>RACE2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>EMSW (Traditional)</td>
<td></td>
</tr>
</tbody>
</table>
Admin

Big Ideas
[MSB-1 Ideas]

- Intuition => looked at canonical FSM case
  - few context can reduce LUT requirements considerably (factor dissimilar logic)
  - similar logic more efficient in local control
  - overall, moderate contexts (e.g. 8)
    - exploits both properties
    - better than extremes
      - single context (all local control)
      - full partition
      - flat memory (except for smallest FSMs)

Big Ideas
[MSB Ideas]

- **Control**: where data affects instructions (operation)
- Two forms:
  - local control
    - all ops resident → fast selection
  - instruction selection
    - may allow us to reduce instantaneous work requirements
    - introduce issues
      - depth, granularity, instruction load time