ESE680-002 (ESE534): Computer Organization

Day 12: February 21, 2007
Compute 2:
Cascades, ALUs, PLAs

Last Time

• LUTs
  – area
  – structure
  – big LUTs vs. small LUTs with interconnect
  – design space
  – optimization

Today

• Cascades
• ALUs
• PLAs

Last Time

• Larger LUTs
  – Less interconnect delay
  + General: Larger compute blocks
  – Minimize interconnect crossings
  - Large LUTs
  – Not efficient for typical logic structure

Different Structure

• How can we have “larger” compute nodes (less general interconnect) without paying huge area penalty of large LUTs?

Structure in subgraphs

• Small LUTs capture structure
• What structure does a small-LUT-mapped netlist have?
Structure

• LUT sequences ubiquitous

Hardwired Logic Blocks

Single Output

Hardwired Logic Blocks

Two outputs

Delay Model

• $T_{cascade} = T_{(3LUT)} + T_{(mux)}$
• Don’t pay
  – General interconnect
  – Full 4-LUT delay

Options

Chung & Rose Study

Programmable Interconnect

Hard-Wired Link

Figure 8: Delay Study HLB Topologies

[Chung & Rose, DAC ’92]
### Cascade LUT Mappings

<table>
<thead>
<tr>
<th>Logic Block</th>
<th>W_L</th>
<th>W_H</th>
<th>% Area</th>
<th>% Area</th>
<th>% Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUT 4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LUT 8</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>LUT 16</td>
<td>16</td>
<td>16</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>LUT 32</td>
<td>32</td>
<td>32</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

[Chung & Rose, DAC '92]

### ALU vs. Cascaded LUT?

![ALU vs. LUT Cascade Diagram]

**Datapath Cascade**

- **ALU/LUT (datapath) Cascade**
  - Long "serial" path w/out general interconnect
  - Pay only Tmux and nearest-neighbor interconnect

### 4-LUT Cascade ALU

![4-LUT Cascade ALU Diagram]

**ALU vs. LUT?**

- Compare/contrast
- **ALU**
  - Only subset of ops available
  - Denser coding for those ops
  - Smaller
  - ...but interconnect dominates
  - [Datapath width orthogonal to function]

### Parallel Prefix LUT Cascade?

- Can we do better than \(N \times Tmux\)?
- Can we compute LUT cascade in \(O(\log(N))\) time?
- Can we compute mux cascade using parallel prefix?

- Can we make mux cascade associative?
Parallel Prefix Mux cascade

• How can mux transform $S \rightarrow \text{mux-out}$?
  - $A=0$, $B=0 \rightarrow \text{mux-out}=0$
  - $A=1$, $B=1 \rightarrow \text{mux-out}=1$
  - $A=0$, $B=1 \rightarrow \text{mux-out}=S$
  - $A=1$, $B=0 \rightarrow \text{mux-out}=/S$

Parallel Prefix Mux cascade

• How can 2 muxes transform input?
• Can I compute 2-mux transforms from 1 mux transforms?

Two-mux transforms

- $SS \rightarrow S$
- $SG \rightarrow G$
- $SB \rightarrow S$
- $SI \rightarrow I$
- $GS \rightarrow S$
- $GG \rightarrow G$
- $GB \rightarrow G$
- $GI \rightarrow G$
- $BS \rightarrow S$
- $GB \rightarrow G$
- $BB \rightarrow B$
- $IB \rightarrow I$
- $SI \rightarrow S$
- $GI \rightarrow G$
- $BI \rightarrow I$
- $II \rightarrow B$

Generalizing mux-cascade

• How can $N$ muxes transform the input?
• Is mux transform composition associative?

Parallel Prefix Mux-cascade

Can be hardwired, no general interconnect
“ALU”s Unpacked

Traditional/Datapath ALUs
1. SIMD/Datapath Control
   • Architecture variable w
2. Long Cascade
   • Typically also w, but can shorter/longer
   • Amenable to parallel prefix implementation in O(log(w)) time w/ O(w) space
3. Restricted function
   • Reduces instruction bits
   • Reduces expressiveness

Commercial Devices

Xilinx XC4000 CLB

Xilinx Virtex-II

Figure 14: Virtex-II CLB Element
**PLA**

- Directly implement flat (two-level) logic
  
  \[ O = a \cdot b \cdot c \cdot d + !a \cdot b \cdot !d + b \cdot !c \cdot d \]

- Exploit substrate properties allow wired-OR

---

**Wired-or**

- Connect series of inputs to wire
- Any of the inputs can drive the wire high

---

**Wired-or**

- Implementation with Transistors

---

**Programmable Wired-or**

- Use some memory function to programmable connect (disconnect) wires to OR
- Fuse:

---

**Diagram Wired-or**

- Gate-memory model
Wired-or array

- Build into array
  - Compute many different or functions from set of inputs

Combined or-arrays to PLA

- Combine two or (nor) arrays to produce PLA (and-or array)

PLA

- Can implement each and on single line in first array
- Can implement each or on single line in second array

PLA Product Terms

- Can be exponential in number of inputs
- E.g. n-input xor (parity function)
  - When flatten to two-level logic, requires exponential product terms
  - \( a*b + !a*b \)
  - \( a*b*c + !a*b*c + !a*b*!c + a*b*!c \)
- …and shows up in important functions
  - Like addition…

PLAs

- Fast Implementations for large ANDs or ORs
- Number of P-terms can be exponential in number of input bits
  - most complicated functions
  - not exponential for many functions
- Can use arrays of small PLAs
  - to exploit structure
  - like we saw arrays of small memories last time
PLAs vs. LUTs?

- Look at Inputs, Outputs, P-Terms
  - minimum area (one study, see paper)
  - K=10, N=12, M=3
- A(PLA 10,12,3) comparable to 4-LUT?
  - 80-130%?
  - 300% on ECC (structure LUT can exploit)
- Delay?
  - Claim 40% fewer logic levels (4-LUT)
  - (general interconnect crossings)

[Kouloheris & El Gamal/CICC’92]
Big Ideas
[MSB Ideas]
- Programmable Interconnect allows us to exploit that structure
  - want to match to application structure
  - Prog. interconnect delay expensive
- Hardwired Cascades
  - key technique to reducing delay in programmables
- PLAs
  - canonical two level structure
  - hardwire portions to get Memories, PALs

Big Ideas
[MSB-1 Ideas]
- Better structure match with hardwired LUT cascades