ESE534: Computer Organization

Compute 2:
Cascades, ALUs, PLAs

Last Time

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

Today

• ALUs
• Cascades
• PLAs

Last Time

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

Preclass

• How does addition delay compare between ALU-based and LUT-based architectures?
  • What is the source of the advantage?

Preclass

• Advantages of ALU bitslice compute block over LUT?
  • Disadvantages?
Implications from LUT-size study

- ALU has more restricted logic functions
  - Will require more blocks

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

Delay Model

- $T_{cascade} = T(3\text{LUT}) + 2T(\text{mux})$
- $T_{cascade} = T(3\text{LUT}) + 2T(\text{mux})$
- Don’t pay
  - General interconnect
  - Full 4-LUT delay
Options

- Ripple adder?
- Associative reduce tree?
- Multiplier?

Chung & Rose Study

- Which most help:

Cascade LUT Mappings

Energy Impact?

- What's the likely energy impact?

ALU vs. Cascaded LUT?
Datapath Cascade

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

ALU vs. LUT?

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

Accelerating LUT Cascade?

- Know can compute addition in $O(\log(N))$ time
- Can we do better than $N \times \text{Tmux}$ for LUT cascade?
- 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 $S \rightarrow \text{mux-out}$?
  - $A=0$, $B=0 \rightarrow \text{mux-out}=0$
  - $A=1$, $B=1 \rightarrow \text{mux-out}=1$
  - $A=0$, $B=1 \rightarrow \text{mux-out}=S$
  - $A=1$, $B=0 \rightarrow \text{mux-out}=\overline{S}$
Parallel Prefix Mux cascade

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

Two-mux transforms

- SS → S
- SG → G
- SB → S
- SI → G
- GS → S
- GG → G
- BB → B
- BI → I
- IS → S
- IG → G
- IB → I
- II → 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

LUT Cascade Implications

- We can compute any LUT cascade in logarithmic time
  - Not just addition carry cascade
  - Can be different functions in each LUT
    - Not demand SIMD op

“ALU”s Unpacked

Traditional/Datapath ALUs combine 3 ideas
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

Virtex 6 & 7 Carry

- Prop = A xor B
- Otherwise
  - Generate
  - Squash
- Sum A xor B xor C

Virtex 6

- V7 similar

Programmable Array Logic (PLAs)

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:

**Programmable Wired-or**

- Gate-memory model

**Diagram Wired-or**

**Wired-or array**

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

**Preclass 6**

- What function (MSP)?
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

Efficiency questions:
- Each and/or is linear in total number of potential inputs (not actual)
- How many product terms between arrays?

Carries

- $C[1] = A[0] \cdot B[0]$
  - $P[1] = 1$
- $P[w] = 2^w - 1$

Sum

- $S[w] = A[w] \oplus B[w] \oplus C[w]$
- $S[w] = (A[w] \cdot B[w]) + A[w] \cdot B[w] \cdot C[w]$
  + $(A[w] \cdot B[w]) + A[w] \cdot B[w] \cdot C[w]$
- Product terms $= 2 \cdot P[w] + 2 \cdot P[terms] / C[w]$
S[2] in MSP

\[
a_{2}^{2}a_{0}^{0}b_{2}^{2}b_{1}^{1} +
a_{2}^{2}a_{1}^{1}a_{0}^{0}b_{2}^{2}b_{1}^{1} +
a_{2}^{2}a_{1}^{1}a_{0}^{0}b_{2}^{2}b_{1}^{1} +
a_{2}^{2}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}a_{0}^{0}b_{2}^{2}b_{1}^{1} +
a_{2}^{2}a_{1}^{1}a_{0}^{0}b_{2}^{2}b_{1}^{1} +
a_{2}^{2}a_{1}^{1}a_{0}^{0}b_{2}^{2}b_{1}^{1} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +
a_{2}^{2}a_{1}^{1}b_{2}^{2}b_{1}^{1}b_{0}^{0} +\]

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
  – \(a^*b^*+a^*b\)
  – \(a^*b^*c+a^*b^*c+a^*b^*c\)
  …and additions, as we just saw…

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)
    • Not comparing to hardwired cascades
    [Kouloheris & El Gamal/CICC’92]
**PLA and PAL**

PAL = Programmable Array Logic

**Conventional/Commercial FPGA**

Alterna 9K (from databook)

**Conventional/Commercial FPGA**

Like PAL

**Admin**

- HW6.1-2 due Friday
- Spring Break next week
- Reading for Monday after break on web
- HW6.3-4 on Wed. after break

**Big Ideas**

[MSB Ideas]

- Programmable Interconnect allows us to exploit structure in logic
  - want to match to application structure
  - Prog. interconnect delay expensive
- Hardwired Cascades
  - key technique to reducing delay in programmables (both ALUs and LUTs)
- PLAs
  - canonical two level structure
  - hardwire portions to get Memories, PALs