ESE534: Computer Organization

Day 3: January 25, 2010
Arithmetic

Work preclass exercise

Last Time
• Boolean logic ⇒ computing any finite function
• Saw gates…and a few properties of logic

Today
• Addition
  – organization
  – design space
  – parallel prefix

Why?
• Start getting a handle on
  – Complexity
  • Area and time
  • Area-time tradeoffs
  – Parallelism
  – Regularity
• Arithmetic underlies much computation
  – grounds out complexity

Preclass

Circuit 1
• Can the delay be reduced?
• How?
• To what?
Tree Reduce AND

• Can the delay be reduced?

Circuit 2

• Can the delay be reduced?

Circuit 3

• Can the delay be reduced?

Brute Force Multi-Output AND

• How big?
  • ~38 here
  • in general about $N^2/2$

Brute Force Multi-Output AND

• Can we do better?

Circuit 4

• Can the delay be reduced?
Addition

Example: Bit Level Addition

- Addition
  - Base 2 example

C: 11011010000
A: 01101101010
B: 01100101100
S: 11010010110

Addition Base 2

- \( A = a_{n-1} \cdot 2^{(n-1)} + a_{n-2} \cdot 2^{(n-2)} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 = \sum (a_i \cdot 2^i) \)
- \( S = A + B \)
- What is the function for \( s_i \) (carry)?
  - \( s_i = \text{carry}_i \oplus a_i \oplus b_i \)
  - \( \text{carry}_i = (a_{i-1} + b_{i-1} + \text{carry}_{i-1}) \geq 2 \)
    - \( = a_i \cdot b_{i-1} + a_{i-1} \cdot \text{carry}_{i-1} + b_i \cdot \text{carry}_{i-1} \)
    - \( = \text{MAJ}(a_{i-1}, b_{i-1}, \text{carry}_{i-1}) \)

Ripple Carry Addition

- Shown operation of each bit
- Often convenient to define logic for each bit, then assemble:
  - bit slice

Ripple Carry Analysis

What is area and delay for N-bit RA adder?

- Area: \( O(N) \) [6n]
- Delay: \( O(N) \) [2n]

Can we do better?

- Lower delay?
Important Observation

• Do we have to wait for the carry to show up to begin doing useful work?
  – We do have to know the carry to get the right answer.
  – How many values can the carry take on?

Idea

• Compute both possible values and select correct result when we know the answer

Preliminary Analysis

• Delay(RA) -- Delay Ripple Adder
• Delay(RA(n)) = k*n  [k=2 this example]
• Delay(RA(n)) = 2*(k*n/2) = 2*RA(n/2)
• Delay(P2A) -- Delay Predictive Adder
• Delay(P2A) = DRA(n/2) + D(mux2)
• ...almost half the delay!

Recurse

• If something works once, do it again.
• Use the predictive adder to implement the first half of the addition

Recurse

• If something works once, do it again.
• Use the predictive adder to implement the first half of the addition

• Delay(P4A(n)) =
  Delay(RA(n/4)) + D(mux2) + D(mux2)
• Delay(P4A(n)) = Delay(RA(n/4)) + 2*Delay(mux2)
Recurse
• By now we realize we’ve been using the wrong recursion
  – should be using the Predictive Adder in the recursion
• Delay(PA(n)) = Delay(PA(n/2)) + D(mux2)
• Every time cut in half…?
• How many times cut in half?
• Delay(PA(n))=log₂(n)*D(mux2)+C
  – C = Delay(PA(1))
    • if use FA for PA(1), then C=2

CLA
• Think about each adder bit as a computing a function on the carry in
  – C[i]=g(c[i-1])
  – Particular function f will depend on a[i], b[i]
  – g=f(a,b)

Functions
• What functions can g(c[i-1]) be?
  – g(x)=1
    • a[i]=b[i]=1
  – g(x)=x
    • a[i] xor b[i]=1
  – g(x)=0
    • a[i]=b[i]=0

Maybe better to show this as a specialization:
  g(c) = carry(a=0,b=0,c) = carry(a=1,b=0,c) = carry(a=0,b=1,c) = carry(a=1,b=1,c)

Combining
• Want to combine functions
  – Compute c[i]=g(g_i,g_{i-2}))
  – Compute compose of two functions
• What functions will the compose of two of these functions be?
  – Same as before
    • Propagate, generate, squash
Compose Rules (LSB MSB)

- GG
- GP
- GS
- PG
- PP
- PS
- SG
- SP
- SS

[work on board]

Compose Rules (LSB MSB)

- GG = G
- GP = G
- GS = S
- PG = G
- PP = P
- PS = S
- SG = G
- SP = S
- SS = S

Combining

- Do it again...
- Combine g[i-3,i-2] and g[i-1,i]
- What do we get?

Reduce Tree

- $S_q = A^*B$
- $Gen = A^*B$

- $S_{out} = S_q + Gen_1^*S_q_0$
- $Gen_{out} = Gen_1 + S_q + Gen_0$

- Delay and Area?
Reduce Tree

- $\text{Sq} = A \cdot B$
- $\text{Gen} = A \cdot B$
- $\text{Sq}_{\text{out}} = \text{Sq} + C \cdot \text{Gen}_{\text{out}}$
- $\text{Gen}_{\text{out}} = \text{Gen} + C \cdot \text{Sq}_{\text{out}}$
- $A(\text{Encode}) = 2$
- $A(\text{Combine}) = 4$
- $A(\text{Carry}) = 2$
- $D(\text{Encode}) = 1$
- $D(\text{Combine}) = 2$
- $D(\text{Carry}) = 1$

Reduce Tree: Delay?

$\text{Delay} = 1 + 2 \log_2(N) + 1$

Reduce Tree: Area?

Area $= 2N + 4(N - 1) + 2$

Reduce Tree: Area & Delay

- Area$(N) = 6N - 2$
- Delay$(N) = 2 \log_2(N) + 2$

Intermediates

- Can we compute intermediates efficiently?
Prefix Tree

- Share terms
- Reduce computes spans
  - 0:3, 4:5
- Reverse tree
  - Combine spans for missing 0:i (i<N)
    - E.g. 0:3+4:5→0:5
- Same size as reduce tree

Intermediate

- Share common terms

Parallel Prefix

Area and Delay?

- Roughly twice the area/delay
- Area= 2N+4N+4N+2N = 10N
- Delay = 4log₂(N)+2

Parallel Prefix

- Important Pattern
- Applicable any time operation is associative
  - Or can be made assoc. as in MAJ case
- Examples of associative functions?
  - Non-associative?
  - Function Composition is always associative
Note: Constants Matter

- Watch the constants
- Asymptotically this Carry-Lookahead Adder (CLA) is great
- For small adders can be smaller with
  - fast ripple carry
  - larger combining than 2-ary tree
  - mix of techniques
- ...will depend on the technology primitives and cost functions

Two's Complement

- positive numbers in binary
- negative numbers
  - subtract 1 and invert
  - (or invert and add 1)

Two's Complement

- 2 = 010
- 1 = 001
- 0 = 000
- -1 = 111
- -2 = 110

Addition of Negative Numbers?

- ...just works

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>111</td>
</tr>
<tr>
<td>B</td>
<td>001</td>
</tr>
<tr>
<td>S</td>
<td>000</td>
</tr>
</tbody>
</table>

- A: 111
- B: 001
- S: 000

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td>010</td>
</tr>
<tr>
<td>S</td>
<td>111</td>
</tr>
</tbody>
</table>

- A: 111
- B: 010
- S: 111

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>111</td>
</tr>
<tr>
<td>B</td>
<td>110</td>
</tr>
<tr>
<td>S</td>
<td>101</td>
</tr>
</tbody>
</table>

- A: 111
- B: 110
- S: 101

Subtraction

- Negate the subtracted input and use adder
  - which is:
    - invert input and add 1
    - works for both positive and negative input
      - -001 \(\rightarrow\) 110 + 1 = 111
      - -111 \(\rightarrow\) 000 + 1 = 001
      - -000 \(\rightarrow\) 111 + 1 = 000
      - -010 \(\rightarrow\) 101 + 1 = 110
      - -110 \(\rightarrow\) 001 + 1 = 010

Subtraction (add/sub)

- Note: you can use the "unused" carry input at the LSB to perform the "add 1"
Overflow?

\[
\begin{align*}
A: & \quad 111 \quad A: \quad 110 \quad A: \quad 111 \quad A: \quad 111 \\
B: & \quad 001 \quad B: \quad 001 \quad B: \quad 010 \quad B: \quad 110 \\
S: & \quad 000 \quad S: \quad 111 \quad S: \quad 001 \quad S: \quad 101 \\
\end{align*}
\]

\[
\begin{align*}
A: & \quad 001 \quad A: \quad 011 \quad A: \quad 111 \\
B: & \quad 001 \quad B: \quad 001 \quad B: \quad 100 \\
S: & \quad 010 \quad S: \quad 100 \quad S: \quad 011 \\
\end{align*}
\]

Overflow\(= (A.s == B.s) \times (A.s! = S.s)\)

Admin

- Office Hours: W 2pm

Big Ideas

[MSB Ideas]

- Can build arithmetic out of logic

Big Ideas

[MSB-1 Ideas]

- Associativity
- Parallel Prefix
- Can perform addition
  - in log time
  - with linear area