More Timing Analysis

Binary Arithmetic
Announcements

• HW 4 due tomorrow

• HW 5 is posted

• Prelab 3A & Lab 2 report due next week
  – Form group before the submission
Review: Important Timing Parameters

- $t_{setup}$
- $t_{hold}$
- $t_{comb}$
- $t_{ffpd}$
- $t_{clk}$
- $t_H$
- $t_L$
Timing Analysis Discussion (1)

• To achieve a higher clock frequency (i.e., smaller clock period), would you prefer

  1) A smaller combinational delay or a larger one?

  2) A smaller setup time or a larger one?

  3) A smaller hold time or a larger one?

  4) A negative clock skew or a positive one?
     • A smaller skew or a larger one?
Example: Setup Time Analysis

- Assumptions:
  1. Uniform gate delay = 1ns
  2. FF propagation delay = 1ns
  3. Setup time = 2ns, hold time = 1ns

- What’s the best achievable cycle time?

\[
t_{clk} \geq t_{ffpd(\text{max})} + t_{comb(\text{max})} + t_{\text{setup}} = 1 + 3 + 2 = 6\text{ns}
\]
Review: Negative Clock Skew

Sending FF receives clock later than receiving FF

\[ t_{\text{ffpd}}(\text{max}) + t_{\text{comb}}(\text{max}) + t_{\text{setup}} \leq t_{\text{clk}} - t_{\text{skew}}(\text{max}) \]

Harmful skew for meeting setup time constraint
Review: Positive Clock Skew

Receiving FF receives clock later than sending FF

\[ t_{\text{ffpd}(\text{max})} + t_{\text{comb}(\text{max})} + t_{\text{setup}} \leq t_{\text{clk}} + t_{\text{skew(min)}} \]

Beneficial skew for meeting setup time constraint
Timing Analysis Discussion (2)

• To **avoid hold time violation**, would you prefer

  1) **A smaller combinational delay** or a larger one?

  2) **A smaller setup time** or a larger one?

  3) **A smaller hold time** or a larger one?

  4) **A negative clock skew** or a positive one?
     • **A smaller skew** or a larger one?
Hold Time With Positive Clock Skew

Receiving FF receives clock later than sending FF

\[ t_{\text{ffpd(min)}} + t_{\text{comb(min)}} \geq t_{\text{hold}} + t_{\text{skew(max)}} \]

Harmful skew for meeting hold time constraint
Hold Time With Negative Clock Skew

What if sending FF receives clock later than receiving FF?

\[ t_{ffpd}(\text{min}) + t_{comb}(\text{min}) \geq t_{hold} - t_{skew}(\text{min}) \]

Beneficial skew for meeting hold time constraint
Example: Hold Time Analysis with Clock Skew

Clock may arrive at FF2 up to 3ns later than FF1

<table>
<thead>
<tr>
<th></th>
<th>Prop Delay (ns)</th>
<th>Setup</th>
<th>Hold Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>Time (ns)</td>
</tr>
<tr>
<td>FF</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Comb</td>
<td>3</td>
<td>7</td>
<td>-</td>
</tr>
</tbody>
</table>

- Hold time at FF2 met?
Example: Hold Time Analysis with Clock Skew

Clock may arrive at FF2 up to 3ns later than FF1

<table>
<thead>
<tr>
<th></th>
<th>Prop Delay (ns)</th>
<th>Setup Time (ns)</th>
<th>Hold Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td></td>
</tr>
<tr>
<td>FF</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Comb</td>
<td>3</td>
<td>7</td>
<td>-</td>
</tr>
</tbody>
</table>

• Hold time at FF2 met?
  
  \[ t_{\text{ffpd(min)}} + t_{\text{comb(min)}} \geq t_{\text{hold}} + t_{\text{skew(max)}} \]
  
  \[ 1 + 3 \geq 1 + 3 \]

  The hold time constraint is met
Course Content

- Binary numbers and logic gates
- Boolean algebra and combinational logic
- Sequential logic and state machines
- Binary arithmetic
- Memories
- Instruction set architecture
- Processor organization
- Caches and virtual memory
- Input/output
Unsigned Binary Integers

- An $n$-bit unsigned number represents $2^n$ integer values
  - From 0 to $2^n-1$

<table>
<thead>
<tr>
<th></th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
Unsigned Binary Fractions

• For the binary number \( b_{n-1}b_{n-2}\ldots b_1b_0.b_{-1}b_{-2}\ldots b_{-m} \) the decimal number is

\[
D = \sum_{i=-m}^{n-1} b_i \cdot 2^i
\]

• Examples

\( 101.001_2 = ? \)
Unsigned Binary Addition

- **Just like base-10**
  - Add from right to left, propagating carry

\[
\begin{align*}
10010 \quad (18) & \quad + \quad 01001 \quad (9) & \quad + \quad 01011 \quad (11) & \quad + \quad 00011 \quad (3) \\
& \quad \quad (27) & \quad \quad (29) & \quad \quad 10010 \quad (18)
\end{align*}
\]
Signed Magnitude Representation

- **Most significant bit is used as a sign bit**
  - Sign bit of 0 for positive \((001 = 1)\)
  - Sign bit of 1 for negative \((101 = -1)\)

- **Range** is from \(-\left(2^{n-1}-1\right)\) to \(\left(2^{n-1}-1\right)\) for an \(n\)-bit number

- Two representations for zero \((+0 \text{ and } -0)\)

- **Does ordinary binary addition still work?**
  
  \[
  \begin{array}{c}
  001 \text{ (1)} \\
  + 101 \text{ (-1)} \\
  \hline
  110 \text{ (not 0)}
  \end{array}
  \]
Another Encoding of Binary Numbers

Wrap-around point
Two’s Complement Representation

- MSB has weight $-2^{n-1}$
- Range of an n-bit number: $-2^{n-1}$ through $2^{n-1}-1$
  - Most negative number ($-2^{n-1}$) has no positive counterpart

<table>
<thead>
<tr>
<th></th>
<th>$-2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Two’s Complement Addition

• Procedure for addition is the same as unsigned addition regardless of the signs of the numbers

\[
\begin{align*}
001 \quad (1) \\
+ 111 \quad (-1) \\
\hline
000 \quad (0)
\end{align*}
\]
Two’s Complement Representation

- Positive numbers and zero are same as unsigned binary representation

- To get two’s complement negative notation of an integer
  - Flip every bit first
  - Then add one

\[ \begin{align*}
001 & \quad \text{(1)} \\
110 \quad \text{(1’s comp)} & \\
+ & \quad 1 \\
111 & \quad \text{(-1)}
\end{align*} \]

\[ \begin{align*}
01001 & \quad \text{(9)} \\
10110 \quad \text{(1’s comp)} & \\
+ & \quad 1 \\
10111 & \quad \text{(-9)}
\end{align*} \]
Two’s Complement (2’s C) Shortcut

- **To get -X**
  - Copy bits from right to left up to and including the first “1”
  - Flip remaining bits to the left

\[
\begin{array}{c}
\text{011010000} \\
\text{100101111} \quad \text{(1’s comp)} \\
+ \quad 1 \\
\text{100110000}
\end{array}
\]

\[
\begin{array}{c}
\text{011010000} \\
\text{(copy)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{100110000} \\
\text{(flip)}
\end{array}
\]
Converting Binary (2’s C) to Decimal

1. If MSB = 1, take two’s complement to get a positive number
2. Add powers of 2 for bit positions that have a “1”
3. If original number was negative, add a minus sign

\[
\begin{align*}
X &= 11100110_{\text{two}} \\
-X &= 00011010 \\
&= 2^4 + 2^3 + 2^1 = 16 + 8 + 2 \\
&= 26_{\text{ten}} \\
X &= -26_{\text{ten}}
\end{align*}
\]

Assuming 8-bit 2’s complement numbers

\[\begin{array}{c|c}
 n & 2^n \\
\hline
0 & 1 \\
1 & 2 \\
2 & 4 \\
3 & 8 \\
4 & 16 \\
5 & 32 \\
6 & 64 \\
7 & 128 \\
8 & 256 \\
9 & 512 \\
10 & 1024 \\
\end{array}\]
Converting Decimal to Binary (2’s C)

First Method: *Division*

1. Change to nonnegative decimal number
2. Divide by two – remainder is least significant bit
3. Keep dividing by two until answer is zero, recording remainders from right (LSB) to left
4. **Append a zero as the MSB;**
   if original number X was negative, return X’+1

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 104_{ten}</td>
<td>104/2 = 52 r0</td>
<td>bit 0 = 0</td>
</tr>
<tr>
<td></td>
<td>52/2 = 26 r0</td>
<td>bit 1 = 0</td>
</tr>
<tr>
<td></td>
<td>26/2 = 13 r0</td>
<td>bit 2 = 0</td>
</tr>
<tr>
<td></td>
<td>13/2 = 6 r1</td>
<td>bit 3 = 1</td>
</tr>
<tr>
<td></td>
<td>6/2 = 3 r0</td>
<td>bit 4 = 0</td>
</tr>
<tr>
<td></td>
<td>3/2 = 1 r1</td>
<td>bit 5 = 1</td>
</tr>
<tr>
<td></td>
<td>1/2 = 0 r1</td>
<td>bit 6 = 1</td>
</tr>
</tbody>
</table>

X = 01101000_{two}
Converting Decimal to Binary (2’s C)

Second Method: Subtract Powers of Two

1. Change to nonnegative decimal number
2. Subtract largest power of two less than or equal to number
3. Put a one in the corresponding bit position
4. Keep subtracting until result is zero
5. Append a zero as MSB; if original was X negative, return X’+1

\[
\begin{array}{l}
X = 104_{\text{ten}} \\
\quad 104 - 64 = 40 \quad \text{bit 6 = 1} \\
\quad 40 - 32 = 8 \quad \text{bit 5 = 1} \\
\quad 8 - 8 = 0 \quad \text{bit 3 = 1} \\
X = 01101000_{\text{two}}
\end{array}
\]
Sign Extension

- Replicate the MSB (sign bit)
  
<table>
<thead>
<tr>
<th>4-bit</th>
<th>8-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100 (4)</td>
<td>00000100 (still 4)</td>
</tr>
<tr>
<td>1100 (-4)</td>
<td>11111100 (still -4)</td>
</tr>
</tbody>
</table>

- Necessary for adding a two’s complement numbers of different lengths
Fixed Size Representation

- Microprocessors usually represent numbers as fixed size $n$-bit values

- Result of adding two $n$-bit integers is stored as $n$ bits

- Integers are typically 32 or 64 bits (words)
  - 4 or 8 bytes
  - byte = 8 bits
Fixed Size Addition

- Examples with $n = 4$

\[
\begin{align*}
2 & \quad 0010 \\
+ & \quad 3 \quad 0011 \\
& \quad 5 \quad 0101 \\
\hline
5 & \quad 0101
\end{align*}
\]

\[
\begin{align*}
2 & \quad 0010 \\
+ & \quad -3 \quad 1101 \\
& \quad -1 \quad 1111 \\
\hline
4 & \quad 0100
\end{align*}
\]

\[
\begin{align*}
-2 & \quad 1110 \\
+ & \quad -6 \quad 1010 \\
& \quad -8 \quad 1000 \\
\hline
-3 & \quad 1101
\end{align*}
\]

\[
\begin{align*}
-7 & \quad 1001 \\
+ & \quad -4 \quad 1100 \\
& \quad 5 \quad 0101 \\
\hline
\end{align*}
\]

Something went wrong!
Overflow

- If operands are too big, sum cannot be represented as \( n \)-bit 2’s complement number

\[
\begin{align*}
01000 & \quad (8) \\
+ 01001 & \quad (9) \\
10001 & \quad (-15)
\end{align*}
\begin{align*}
11000 & \quad (-8) \\
+ 10111 & \quad (-9) \\
01111 & \quad (+15)
\end{align*}
\]

- Overflow occurs if
  - Signs of both operands are the same, and
  - Sign of sum is different

- Another test (easy to do in hardware)
  - Carry into MSB does not equal carry out
Before Next Class

• H&H 5.5

Next Time

More Binary Arithmetic
ALU