Today

- Data Multiplexors
- Arithmetic and Logic Unit
- Adder/Subtractor
- Programmable Logic Arrays

Review

- Pipeline big-delay CL for faster clock
- Finite State Machines extremely useful
  - You’ll see them again in 150, 152 & 164
- Use this table and techniques we learned to transform from 1 to another

Data Multiplexor (here 2-to-1, n-bit-wide)

N instances of 1-bit-wide mux

How many rows in TT?

How do we build a 1-bit-wide mux?

\[ c = \overline{a}b + a\overline{b} + \overline{a}\overline{b} + a\overline{b} = \overline{s}(\overline{a}b + a\overline{b}) + s(ab + \overline{a}b) = \overline{s}(a\overline{b} + b) + s((\overline{a} + a)\overline{b}) = \overline{s}(a1) + s(1b) = \overline{sa} + sb \]
4-to-1 Multiplexor?

How many rows in TT?

\[ e = \overline{s_1 s_0} a + \overline{s_1 s_0} b + \overline{s_1 s_0} c + s_1 s_0 d \]

Is there any other way to do it?

Hint: NCAA tourney!

Ans: Hierarchically!

Do you really understand NORs?

• If one input is 1, what is a NOR?
• If one input is 0, what is a NOR?

A  B  NOR
0  0  1
0  1  0
1  0  0
1  1  0

NOR

A  B  NOR
0  0  B'
0  1  1
1  0  1
1  1  0

Do you really understand NANDs?

• If one input is 1, what is a NAND?
• If one input is 0, what is a NAND?

A  B  NAND
0  0  1
0  1  1
1  0  1
1  1  0

NAND

A  B  NAND
0  0  1
0  1  1
1  0  1
1  1  0

What does it mean to “clobber” midterm?

• You STILL have to take the final even if you aced the midterm!
• The final will contain midterm-material Qs and new, post-midterm Qs
• They will be graded separately
• If you do “better” on the midterm-material, we will clobber your midterm with the “new” score! If you do worse, midterm unchanged.
• What does “better” mean?
  • Better w.r.t. Standard Deviations around mean
• What does “new” mean?
  • Score based on remapping St. Dev. score on final midterm-material to midterm score St. Dev.

“Clobber the midterm” example

• Midterm
  • Mean: 45
  • Standard Deviation: 14
  • You got a 31, one \( \sigma \) below, i.e., mean - \( \sigma \)

• Final Midterm-Material Questions
  • Mean: 40
  • Standard Deviation: 20
  • You got a 60, one \( \sigma \) above
  • Your new midterm score is now mean + \( \sigma \) = 45 + 14 = 59 (~ double your old score)!
Administrivia

• Any administrivia?

Arithmetic and Logic Unit

• Most processors contain a special logic block called “Arithmetic and Logic Unit” (ALU)

• We’ll show you an easy one that does ADD, SUB, bitwise AND, bitwise OR

\[
\begin{align*}
A & \quad B \\
\downarrow & \quad \downarrow \\
\text{ALU} & \\
\downarrow & \quad \downarrow \\
S & \quad R
\end{align*}
\]

when $S=00$, $R=A+B$
when $S=01$, $R=A-B$
when $S=10$, $R=A \text{ AND } B$
when $S=11$, $R=A \text{ OR } B$

Adder/Subtractor Design -- how?

• Truth-table, then determine canonical form, then minimize and implement as we’ve seen before

• Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer

Our simple ALU

Adder/Subtractor -- One-bit adder LSB...

\[
\begin{array}{cccccccc}
 & a_3 & a_2 & a_1 & a_0 & b_3 & b_2 & b_1 & b_0 \\
+ & S_3 & S_2 & S_1 & S_0 \\
\hline
S_0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
S_1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Adder/Subtractor -- One-bit adder (1/2)...

\[
\begin{array}{cccccccc}
a_3 & a_2 & a_1 & a_0 & b_3 & b_2 & b_1 & b_0 \\
+ & S_3 & S_2 & S_1 & S_0 \\
\hline
a & b & c & S & c_{i+1} \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{align*}
S_i &= \text{[Expression]} \\
c_{i+1} &= \text{[Expression]}
\end{align*}
\]
Adder/Subtractor – One-bit adder (2/2)

\[ s_i = \text{XOR}(a_i, b_i, c_i) \]
\[ c_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i \]

What about overflow?

- Consider a 2-bit signed \# & overflow:
  - \(+10 = -2 + -2 \text{ or } -1\)
  - \(+11 = -1 + -2 \text{ only}\)
  - \(+00 = 0 \text{ NOTHING!}\)
  - \(+01 = 1 + 1 \text{ only}\)

- Highest adder
  - \(C_i = \text{Carry-in} = C_{in}\)
  - \(C_{out} = \text{Carry-out} = C_{out}\)
  - No \(C_{out}\) or \(C_{in} = \text{NO overflow!}\)
  - \(C_{in}\) and \(C_{out} = \text{NO overflow!}\)
  - \(C_{in}\) but no \(C_{out} = \text{A,B both > 0, overflow!}\)
  - \(C_{out}\) but no \(C_{in} = \text{A,B both < 0, overflow!}\)

N 1-bit adders \(\Rightarrow\) 1 N-bit adder

What about overflow?

- Consider a 2-bit signed \# & overflow:
  - \(+10 = -2\)
  - \(+11 = -1\)
  - \(+00 = 0\)
  - \(+01 = 1\)

- Overflows when...
  - \(C_{in}\) but no \(C_{out} = \text{A,B both > 0, overflow!}\)
  - \(C_{out}\) but no \(C_{in} = \text{A,B both < 0, overflow!}\)

Overflow = \(C_n \text{ XOR } C_{n-1}\)

Extremely Clever Subtractor

Review: Finite State Machine (FSM)

- States represent possible output values.
- Transitions represent changes between states based on inputs.
- Implement with CL and clocked register feedback.
Finite State Machines extremely useful!

- They define
  - How output signals respond to input signals and previous state.
  - How we change states depending on input signals and previous state.
- We could implement very detailed FSMs with Programmable Logic Arrays.

Taking advantage of sum-of-products

- Since sum-of-products is a convenient notation and way to think about design, offer hardware building blocks that match that notation.
- One example is Programmable Logic Arrays (PLAs).
- Designed so that can select (program) ands, ors, complements after you get the chip.
  - Late in design process, fix errors, figure out what to do later, ...

Programmable Logic Arrays

- Pre-fabricated building block of many AND/OR gates
  - "Programmed" or "Personalized" by making or breaking connections among gates
  - Programmable array block diagram for sum of products form

Enabling Concept

- Shared product terms among outputs

Before Programming

- All possible connections available before "programming"

After Programming

- Unwanted connections are "blown"
  - Fuse (normally connected, break unwanted ones)
  - Anti-fuse (normally disconnected, make wanted connections)
Alternate Representation

- Short-hand notation—don’t have to draw all the wires
- X Signifies a connection is present and perpendicular signal is an input to gate

notation for implementing

\[ F_0 = A B \quad F_3 = C \quad F_5 = T \quad F_7 = T \]

Where \( A \) and \( B \) are the inputs.

Programmable Logic Arrays are often used to implement our CL.

“And In conclusion…”

- Usemuxesto select among input
  - \( S \) input bits selects \( 2^S \) inputs
  - Each input can be \( n \)-bits wide, indep of \( S \)
- Implement muxes hierarchically
- ALU can be implemented using a mux
  - Coupled with basic block elements
- \( N \)-bit adder-subtractor done using \( N \) 1-bit adders with XOR gates on input
  - XOR serves as conditional inverter

Peer Instruction

A. SW can peek at HW (past ISA abstraction boundary) for optimizations
B. SW can depend on particular HW implementation of ISA
C. Timing diagrams serve as a critical debugging tool in the EE toolkit

A. HW feedback akin to SW recursion
B. We can implement a D-Q flipflop and simple CL (And, Or, Not gates)
C. You can build a FSM to signal when an equal number of 0s and 1s has appeared in the input.

Peer Instruction

A. \((a+b)\cdot (\overline{a+b}) = b\)
B. N-input gates can be thought of cascaded 2-input gates. I.e.,
\( (a \cdot \overline{bc} \cdot \overline{d} \cdot e) = \overline{a} \cdot \overline{bc} \cdot \overline{(d \cdot e)} \)
where \( A \) is one of AND, OR, XOR, NAND
C. You can use NOR(s) with clever wiring to simulate AND, OR, & NOT

A. Truth table for mux with 4-bits of signals has \( 2^4 \) rows
B. We could cascade \( N \) 1-bit shifters to make \( 1 \) N-bit shifter for \( s1 \), \( s2 \)
C. If 1-bit adder delay is \( T \), the \( N \)-bit adder delay would also be \( T \)