Review: Representations for CL

Use this diagram and techniques we learned to transform from one to another.

[Diagram showing the relationship between Truth Table, Boolean Expression, and Gate Diagram]
Today

- Data Multiplexors
- Arithmetic and Logic Unit (ALU)
- Adder/Subtractor

Read: “Combinational Logic Blocks”, posted on website.

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

“mux”

when \( s=0 \), \( C=A \)
when \( s=1 \), \( C=B \)

Internally, made up of \( n \) instances of 1-bit wide muxes.
1-bit-wide mux

How many rows in TT?

\[ c = \overline{s}ab + \overline{s}ab + s\overline{a}b + sab \]
\[ = \overline{s}(a\overline{b} + ab) + s(\overline{a}b + ab) \]
\[ = \overline{s}(a(\overline{b} + b)) + s((\overline{a} + a)b) \]
\[ = \overline{s}(a(1) + s((1)b) \]
\[ = \overline{s}a + sb \]

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

**\( \overline{s}a + sb \)**
4-to-1 Multiplexor

when S=00, e=a
when S=01, e=b
when S=10, e=c
when S=11, e=d

How many rows in TT?

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

Is there any other way to do it?

Hint: “single elimination” tournament

Ans: Hierarchically!
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

\[
\begin{align*}
\text{when } S=00, \ R &= A+B \\
\text{when } S=01, \ R &= A-B \\
\text{when } S=10, \ R &= A \text{ AND } B \\
\text{when } S=11, \ R &= A \text{ OR } B
\end{align*}
\]
Our simple ALU

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
Adder/Subtracter – One-bit adder LSB...

\[
\begin{array}{cccc|c}
 a_3 & a_2 & a_1 & a_0 \\
 b_3 & b_2 & b_1 & b_0 \\
 \hline
 s_3 & s_2 & s_1 & s_0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
a_0 & b_0 & s_0 & c_1 \\
0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
\end{array}
\]

\[s_0 =\]
\[c_1 =\]

Adder/Subtracter – One-bit adder (1/2)...

\[
\begin{array}{cccc|c|c}
a_i & b_i & c_i & s_i & c_{i+1} \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[s_i =\]
\[c_{i+1} =\]
Adder/Subtracter – One-bit adder (2/2)...

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

N 1-bit adders ⇒ 1 N-bit adder

What about overflow?
What about overflow detection?

° Overflow never when A and B both have the same sign.

° Adding two positive numbers:
  • Can never result in a carry out of the MSB position
  • A carry into the MSB stage turns the n-bit representation into a negative number ⇒ result is too big and overflow occurred.

° Adding two negative numbers:
  • Always results in a carry out of the MSB position
  • No carry into the MSB stage turns the n-bit representation into a positive number ⇒ overflow

° In either case, if carry in to the MSB stage is different from carry out, then overflow has occurred.

\[
\text{overflow} = c_n \text{ xor } c_{n-1}
\]

Subtraction Based on Adder Circuit:
“And In conclusion…”

° Muxes are used to select among inputs
  • S control bits select among $2^S$ inputs
  • Each input can be n-bits wide, indep. of S

° Muxes with many inputs often implemented hierarchically

° ALU can be implemented using a mux
  • Coupled with basic block elements

° N-bit adder-subtractor done using N 1-bit adders
  • XORs at input serves as conditional inverter for substraction