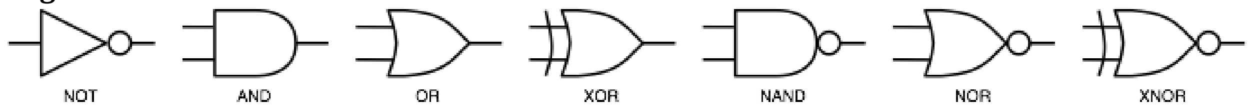
CS 61C
Spring 2010

Combinational Logic & State

Week 9 (3/16)

Michael Greenbaum (cs61c-tf),
adopted from Scott Beamer.

Logic Gates



- Looking at what XNOR does, can you think of another name for it?
  EQUAL, since the output is only high when both inputs are equal.
- How many different two-input logic gates are possible? How many n-input logic gates?
  With an n-bit input, the truth table has 2^n rows. A particular function is an assignment of 0 or 1 to each of these rows. Imagining a function as a 2^n bit number, we count 2^(2^n) total functions, 16 in the case of n=2.
- Build NOT, AND, OR, and XOR using only NAND. To save yourself writing, once you have built a gate, you can re-use it.

NOT – tie input to both inputs of NAND.
AND – NOT on output of NAND.
OR – With inputs A and B, OR is ¬ ((¬A) (¬ B)) (de morgan's law).
XOR – Can write out the canonical form, A(¬B) + (¬A)B, which can be implemented with AND, OR, and NOT.

Pipelining Review

- *Recall*: Minimum clock period = $t_{clk\text{-}to\text{-}q} + t_{CL} + t_{setup}$
- Usually t_{CL} dominates, but it is only the combinational delay between registers
- If we place registers in the critical path, we can shorten the delay by reducing the amount of logic between registers

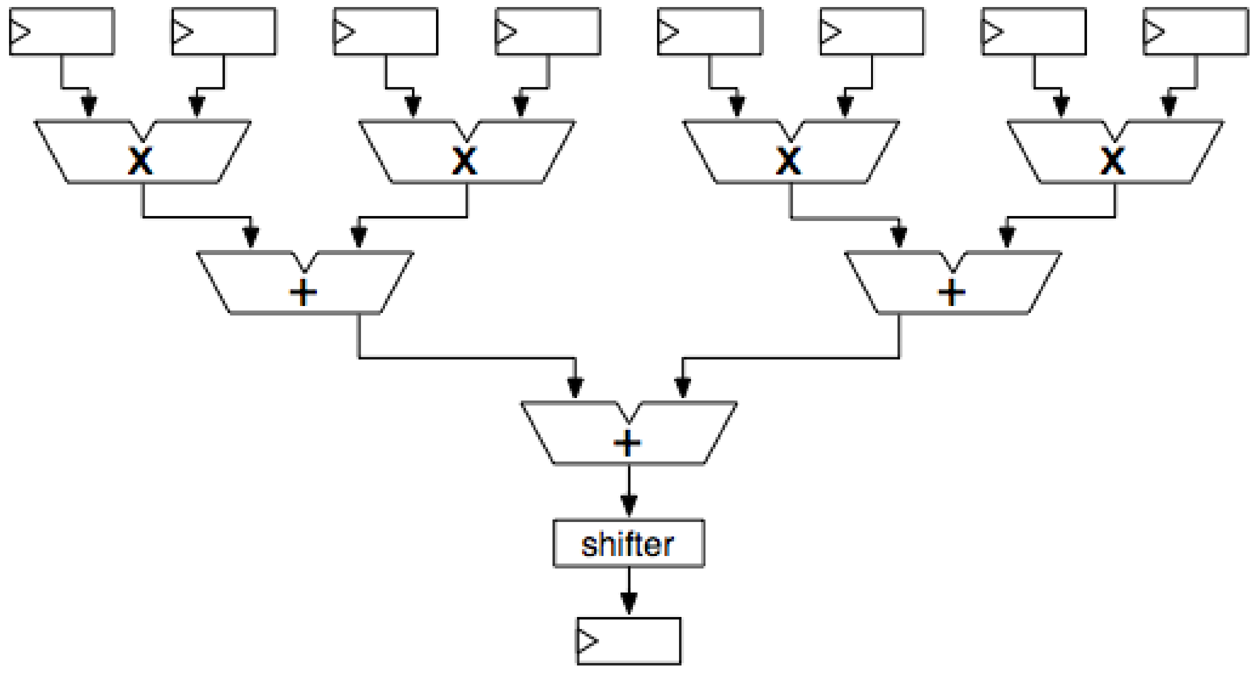
Pipelining Problem

- The circuit below computes the weighted average of 4 values
- Logic Delays - t_{mult} = 55ns, t_{add} = 19ns, t_{shift} = 2ns
- Register Parameters - t_{setup} = 2ns, t_{hold} = 1ns, $t_{clk\text{-}to\text{-}q}$ = 3ns
- What is the critical path delay and the maximum frequency this circuit can operate at?

Critical path – path from a top register to the bottom register.
(3 + 55 + 19 + 19 + 2 + 2) = 100 ns. Max frequency – 10 Mhz (period exactly 100 ns)

- If you add one stage of registers (pipelining), what is the highest frequency you can get?

Best to add registers in a place that minimizes the longest path through the circuit. This would be right after the multiplication. New critical path – (3 + 55 + 2) = 60 ns, highest frequency – 16.67 Mhz.



Boolean Simplification Practice

- Minimize the following boolean expressions:

$(A + B)(A + \overline{B})C$

$\overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + AB\overline{C} +$
$A\overline{B}\overline{C} + ABC + A\overline{B}C$

$\overline{A}B + A\overline{B}$

From left to right: (complemented values in lowercase)

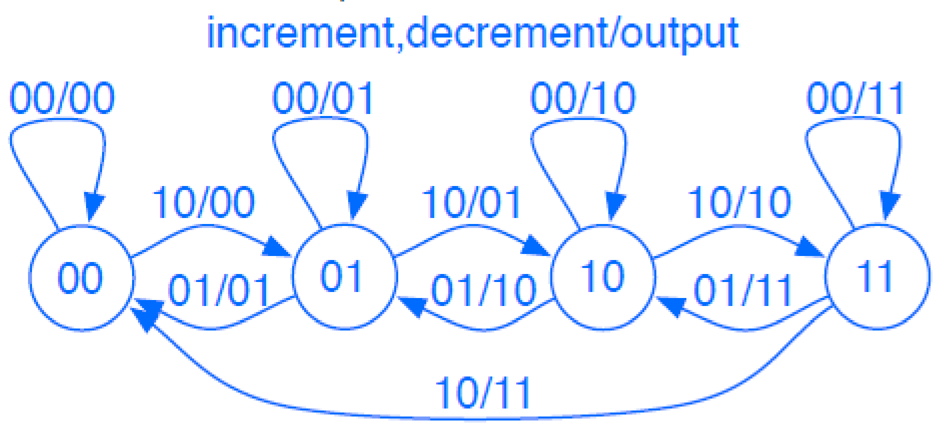
AC

ac (b+B) + AB(c+C) + Ab(c+C)
ac+AB+Ab
ac+A(b+B)
ac+A
(this is not a unique simplification)

A XOR B

Finite State Machine Practice

- *Goal:* A system that can output a value between 0 - 3 with the ability to increment and decrement. This system will have two 1-bit inputs: increment and decrement (as well as clock), and a 2-bit output (the count). If increment is high, the count should increase by one for the next cycle (wrap around if necessary). If decrement is high, the count should decrease by one for the next cycle (wrap around if necessary). If neither is high the system should stay at the same value, and they will never both be high at the same time.
- Draw a finite state machine for this system.



- Assign states binary encodings and complete a truth table for your FSM.

Increment	Decrement	Current State	Next State	Output
0	0	00	00	00
0	1	00	11	00
1	0	00	01	00
1	1	00	xx	00
0	0	01	01	01
0	1	01	00	01
1	0	01	10	01
1	1	01	xx	01
0	0	10	10	10
0	1	10	01	10
1	0	10	11	10
1	1	10	xx	10
0	0	11	11	11
0	1	11	10	11
1	0	11	00	11
1	1	11	xx	11

- Starting from sum-of-product expressions from the truth table, derive simplified expressions for next state as well as the output.

Output = State

$NS0 = CS0 \oplus (I + D)$ $NS1 = CS1\bar{I}\bar{D} + (CS0 \oplus CS1)\bar{I}D + (CS0 \oplus CS1)I\bar{D}$