inst.eecs.berkeley.edu/~cs61c

UC Berkeley CS61C : Machine Structures

Lecture 22 – Representations of Combinatorial Logic Circuits

2007-3-9

TA David "The Punner" Eitan Poll

www.depoll.com

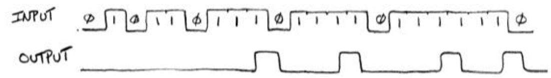
Highly Illogical⇒

I don't have any news for you today, but thought that a Spock reference was pertinent given the topic of this lecture!

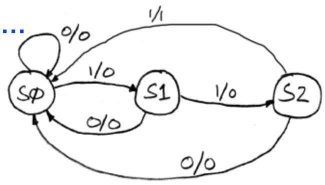
www.startrek.com

Finite State Machine Example: 3 ones…

FSM to detect the occurrence of 3 consecutive 1's in the input.



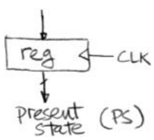
Draw the FSM…



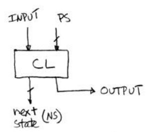
Assume state transitions are controlled by the clock: on each clock cycle the machine checks the inputs and moves to a new state and produces a new output…

Hardware Implementation of FSM

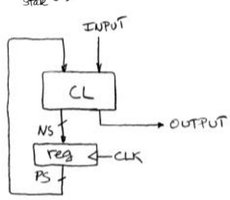
… Therefore a register is needed to hold the a representation of which state the machine is in. Use a unique bit pattern for each state.



\+



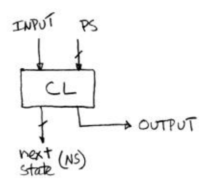
=



Combinational logic circuit is used to implement a function maps from *present state and input* to *next state and output.*

Hardware for FSM: Combinational Logic

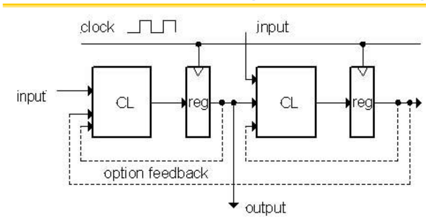
This lecture we will discuss the detailed implementation, but for now can look at its functional specification, truth table form.



Truth table…

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

General Model for Synchronous Systems



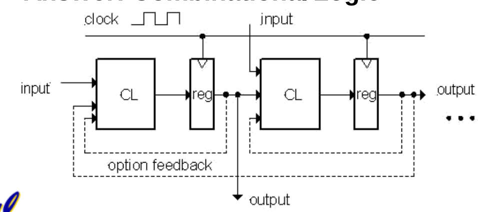
- Collection of CL blocks separated by registers.
- Registers may be back-to-back and CL blocks may be back-to-back.
- Feedback is optional.
- Clock signal(s) connects only to clock input of registers.

Review

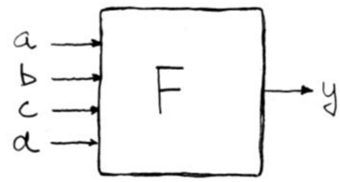
- **State elements are used to:**
  - **Build memories**
  - **Control the flow of information between other state elements and combinational logic**
- **D-flip-flops used to build registers**
- **Clocks tell us when D-flip-flops change**
  - **Setup and Hold times important**
- **We pipeline long-delay CL for faster clock**
- **Finite State Machines extremely useful**
  - **Represent states and transitions**

Combinational Logic

- FSMs had states and transitions
- How to we get from one state to the next?
- Answer: Combinational Logic



Truth Tables

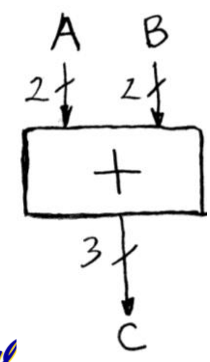


a	b	c	d	y
0	0	0	0	F(0,0,0,0)
0	0	0	1	F(0,0,0,1)
0	0	1	0	F(0,0,1,0)
0	0	1	1	F(0,0,1,1)
0	1	0	0	F(0,1,0,0)
0	1	0	1	F(0,1,0,1)
0	1	1	0	F(0,1,1,0)
0	1	1	1	F(0,1,1,1)
1	0	0	0	F(1,0,0,0)
1	0	0	1	F(1,0,0,1)
1	0	1	0	F(1,0,1,0)
1	0	1	1	F(1,0,1,1)
1	1	0	0	F(1,1,0,0)
1	1	0	1	F(1,1,0,1)
1	1	1	0	F(1,1,1,0)
1	1	1	1	F(1,1,1,1)

TT Example #1: 1 iff one (not both) a,b=1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

TT Example #2: 2-bit adder



A a_1a_0	B b_1b_0	C $c_2c_1c_0$
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

How Many Rows?

TT Example #3: 32-bit unsigned adder

A	B	C
000 ... 0	000 ... 0	000 ... 00
000 ... 0	000 ... 1	000 ... 01
.	.	.
.	.	.
.	.	.
111 ... 1	111 ... 1	111 ... 10

How Many Rows?

TT Example #3: 3-input majority circuit

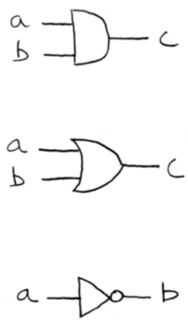
a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Logic Gates (1/2)

AND

OR

NOT



ab	c
00	0
01	0
10	0
11	1

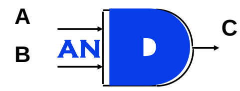
ab	c
00	0
01	1
10	1
11	1

a	b
0	1
1	0

And vs. Or review – Dan's mnemonic

AND Gate

Symbol



Definition

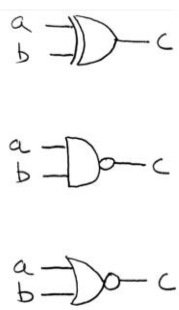
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Logic Gates (2/2)

XOR

NAND

NOR



ab	c
00	0
01	1
10	1
11	0

ab	c
00	1
01	1
10	1
11	0

ab	c
00	1
01	0
10	0
11	0

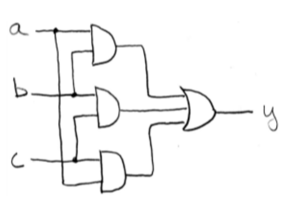
2-input gates extend to n-inputs

- N-input XOR is the only one which isn't so obvious
- It's simple: XOR is a 1 iff the # of 1s at its input is odd ⇒

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

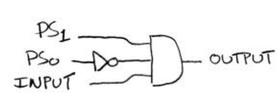
Truth Table ⇒ Gates (e.g., majority circ.)

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

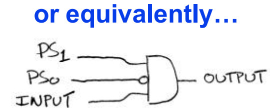


Truth Table ⇒ Gates (e.g., FSM circ.)

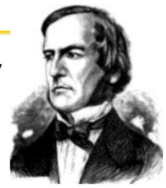
PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1



or equivalently…



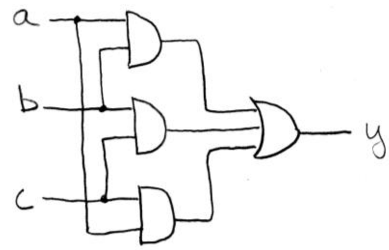
Boolean Algebra



- George Boole, 19th Century mathematician
- Developed a mathematical system (algebra) involving logic
  - later known as "Boolean Algebra"
- Primitive functions: AND, OR and NOT
- The power of BA is there's a one-to-one correspondence between circuits made up of AND, OR and NOT gates and equations in BA

+ means OR,• means AND, $\overline{x}$ means NOT

Boolean Algebra (e.g., for majority fun.)

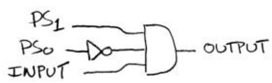


y = a • b + a • c + b • c

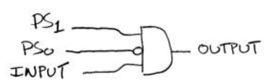
y = ab + ac + bc

Boolean Algebra (e.g., for FSM)

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

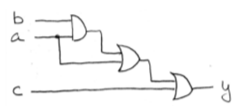


or equivalently…



$y = PS_1 \cdot \overline{PS_0} \cdot INPUT$

BA: Circuit & Algebraic Simplification



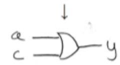
original circuit

$$y = ((ab) + a) + c$$

equation derived from original circuit

$$\begin{aligned} &= ab + a + c \\ &= a(b+1) + c \\ &= a(1) + c \\ &= a + c \end{aligned}$$

algebraic simplification



simplified circuit

BA also great for circuit verification
Circ X = Circ Y?
use BA to prove!

Laws of Boolean Algebra

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	complementarity
$x \cdot 0 = 0$	$x + 1 = 1$	laws of 0's and 1's
$x \cdot 1 = x$	$x + 0 = x$	identities
$x \cdot x = x$	$x + x = x$	idempotent law
$x \cdot y = y \cdot x$	$x + y = y + x$	commutativity
$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$	associativity
$x(y+z) = xy + xz$	$x + yz = (x+y)(x+z)$	distribution
$xy + x = x$	$(x+y)x = x$	uniting theorem
$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{(x+y)} = \overline{x} \cdot \overline{y}$	DeMorgan's Law

Boolean Algebraic Simplification Example

$$\begin{aligned} y &= ab + a + c \\ &= a(b+1) + c && \textit{distribution, identity} \\ &= a(1) + c && \textit{law of 1's} \\ &= a + c && \textit{identity} \end{aligned}$$

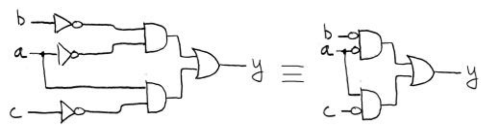
Canonical forms (1/2)

	abc	y
$\overline{a} \cdot \overline{b} \cdot \overline{c}$	000	1
$\overline{a} \cdot \overline{b} \cdot c$	001	1
	010	0
	011	0
$a \cdot \overline{b} \cdot \overline{c}$	100	1
	101	0
$a \cdot b \cdot \overline{c}$	110	1
	111	0

Sum-of-products (ORs of ANDs)

Canonical forms (2/2)

$$
\begin{aligned}
y &= \overline{a}\overline{b}\overline{c} + \overline{a}\overline{b}c + a\overline{b}\overline{c} + ab\overline{c} \\
&= \overline{a}\overline{b}(\overline{c} + c) + a\overline{c}(\overline{b} + b) && \textit{distribution} \\
&= \overline{a}\overline{b}(1) + a\overline{c}(1) && \textit{complementarity} \\
&= \overline{a}\overline{b} + a\overline{c} && \textit{identity}
\end{aligned}
$$



Peer Instruction

A. (a+b)• ($\overline{a}$+b) = b

B. N-input gates can be thought of cascaded 2-input gates. I.e., (a ∆ bc ∆ d ∆ e) = a ∆ (bc ∆ (d ∆ e)) where ∆ is one of AND, OR, XOR, NAND

C. You can use NOR(s) with clever wiring to simulate AND, OR, & NOT

	ABC
1:	FFF
2:	FFT
3:	FTF
4:	FTT
5:	TFF
6:	TFT
7:	TTF
8:	TTT

Peer Instruction Answer (B)

B. N-input gates can be thought of cascaded 2-input gates. I.e., (a ∆ bc ∆ d ∆ e) = a ∆ (bc ∆ (d ∆ e)) where ∆ is one of AND, OR, XOR, NAND…FALSE

Let's confirm!

CORRECT 3-input

XYZ	AND	OR	XOR	NAND
000	0	0	0	1
001	0	1	1	1
010	0	1	1	1
011	0	1	0	1
100	0	1	1	1
101	0	1	0	1
110	0	1	0	1
111	1	1	1	0

CORRECT 2-input

YZ	AND	OR	XOR	NAND
00	0	0	0	1
01	0	1	1	1
10	0	1	1	1
11	1	1	0	0

"And In conclusion…"

- **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**