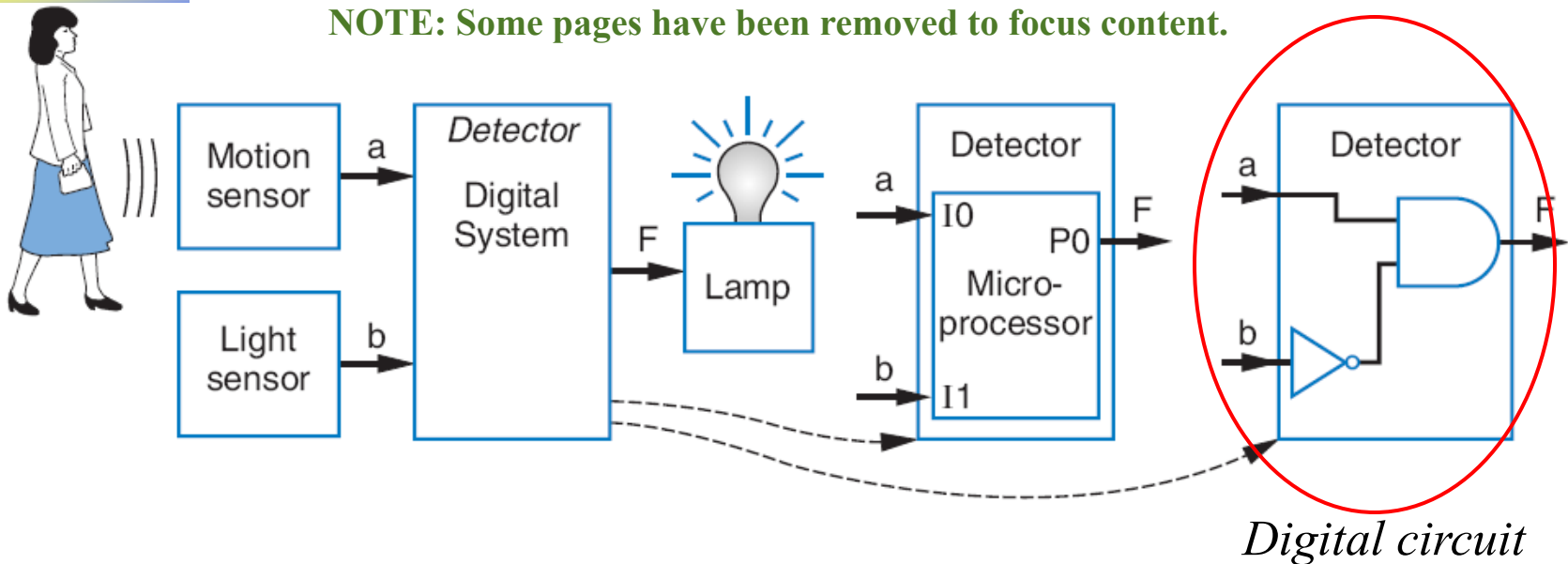


Combinational Circuit Introduction

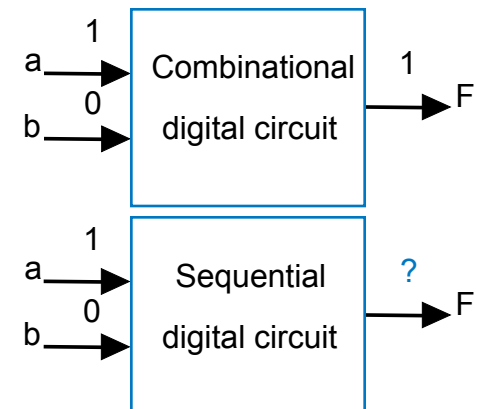
NOTE: Some pages have been removed to focus content.



- We'll start with a simple form of circuit:

- **Combinational circuit**

- A digital circuit whose outputs depend solely on the present combination of the circuit inputs' values
- Built out of simple components: switches and gates



Combinational Logic Design Process

	Step	Description
Step 1	Capture the function	Create a truth table or equations to describe the desired behavior of the combinational logic.
Step 2	Convert to equations	This step is only necessary if you captured the function using a truth table instead of equations. Simplify the equations if desired.
Step 3	Implement as a gate-based circuit	For each output, create a circuit corresponding to the output's equation.

Example: Seat Belt Warning Light System

- Design circuit for warning light
- Sensors
 - $s=1$: seat belt fastened
 - $k=1$: key inserted
 - $p=1$: person in seat
- Capture logic equation
 - What are conditions for warning light to go on?
- Convert equation to circuit



Example: Number of 1s Count

- Problem: Output in binary on two outputs yz the number of 1s on three inputs

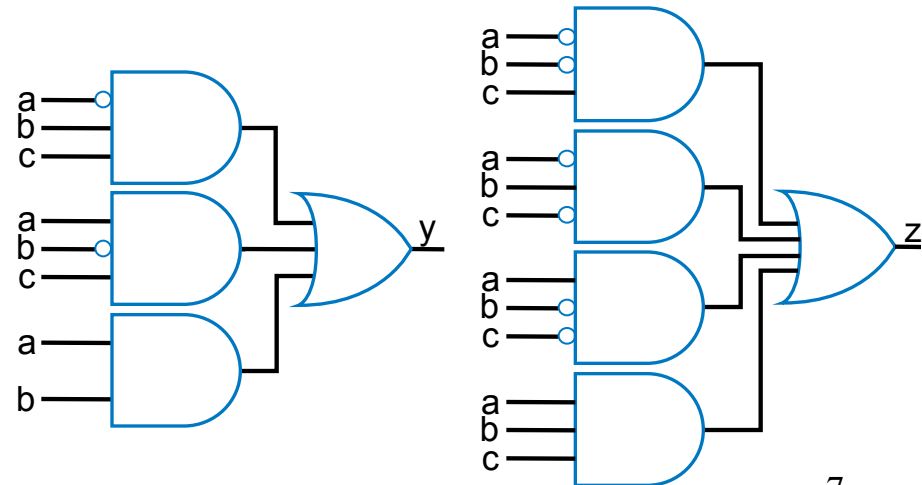
- 010 →
- 101 →
- 000 →

- **Step 1: Capture** the function
 - Truth table or equation?

- **Step 2: Convert** to equation

- **Step 3: Implement** as gates

Inputs			(# of 1s)	Outputs	
a	b	c		y	z
0	0	0	(0)	0	0
0	0	1	(1)	0	1
0	1	0	(1)	0	1
0	1	1	(2)	1	0
1	0	0	(1)	0	1
1	0	1	(2)	1	0
1	1	0	(2)	1	0
1	1	1	(3)	1	1



Sum-of-products canonical forms

- Also known as disjunctive normal form
- Minterm expansion:


					$F = 001 \quad 011 \quad 101 \quad 110 \quad 111$				
					$F = A'B'C + A'BC + AB'C + ABC' + ABC$				
A	B	C	F	F'					
0	0	0	0	1					
0	0	1	1	0					
0	1	0	0	1					
0	1	1	1	0					
1	0	0	0	1					
1	0	1	1	0					
1	1	0	1	0					
1	1	1	1	0					
					$F' = A'B'C' + A'BC' + AB'C'$				

Sum-of-products canonical form (cont'd)

- Product minterm
 - ANDed product of literals – input combination for which output is 1
 - each variable appears exactly once, true or inverted (but not both)

A	B	C	minterms
0	0	0	$A'B'C'$ m0
0	0	1	$A'B'C$ m1
0	1	0	$A'BC'$ m2
0	1	1	$A'BC$ m3
1	0	0	$AB'C'$ m4
1	0	1	$AB'C$ m5
1	1	0	ABC' m6
1	1	1	ABC m7

short-hand notation for
minterms of 3 variables



F in canonical form:

$$\begin{aligned}F(A, B, C) &= \Sigma m(1,3,5,6,7) \\&= m1 + m3 + m5 + m6 + m7 \\&= A'B'C + A'BC + AB'C + ABC' + ABC\end{aligned}$$

canonical form \neq minimal form

$$\begin{aligned}F(A, B, C) &= A'B'C + A'BC + AB'C + ABC + ABC' \\&= (A'B' + A'B + AB' + AB)C + ABC' \\&= ((A' + A)(B' + B))C + ABC' \\&= C + ABC' \\&= ABC' + C \\&= AB + C\end{aligned}$$

Product-of-sums canonical form

- Also known as conjunctive normal form
- Also known as **maxterm** expansion
- Implements “zeros” of a function

					$F =$	000	010	100
					$F =$	$(A + B + C)$	$(A + B' + C)$	$(A' + B + C)$
A	B	C	F	F'				
0	0	0	0	1				
0	0	1	1	0				
0	1	0	0	1				
0	1	1	1	0				
1	0	0	0	1				
1	0	1	1	0				
1	1	0	1	0				
1	1	1	1	0				

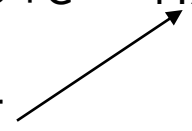
$$F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')$$

Product-of-sums canonical form (cont'd)

- Sum term (or maxterm)
 - ORed sum of literals – input combination for which output is false
 - each variable appears exactly once, true or inverted (but not both)

A	B	C	maxterms	
0	0	0	$A+B+C$	M0
0	0	1	$A+B+C'$	M1
0	1	0	$A+B'+C$	M2
0	1	1	$A+B'+C'$	M3
1	0	0	$A'+B+C$	M4
1	0	1	$A'+B+C'$	M5
1	1	0	$A'+B'+C$	M6
1	1	1	$A'+B'+C'$	M7

short-hand notation for
maxterms of 3 variables



F in canonical form:

$$\begin{aligned}F(A, B, C) &= \prod M(0,2,4) \\&= M0 \bullet M2 \bullet M4 \\&= (A + B + C) (A + B' + C) (A' + B + C)\end{aligned}$$

canonical form \neq minimal form

$$\begin{aligned}F(A, B, C) &= (A + B + C) (A + B' + C) (A' + B + C) \\&= (A + B + C) (A + B' + C) \\&\quad (A + B + C) (A' + B + C) \\&= (A + C) (B + C)\end{aligned}$$

Mapping between canonical forms

- Minterm to maxterm conversion
 - use maxterms whose indices do not appear in minterm expansion
 - e.g., $F(A,B,C) = \sum m(1,3,5,6,7) = \prod M(0,2,4)$
- Maxterm to minterm conversion
 - use minterms whose indices do not appear in maxterm expansion
 - e.g., $F(A,B,C) = \prod M(0,2,4) = \sum m(1,3,5,6,7)$
- Minterm expansion of F to minterm expansion of F'
 - use minterms whose indices do not appear
 - e.g., $F(A,B,C) = \sum m(1,3,5,6,7) \quad F'(A,B,C) = \sum m(0,2,4)$
- Maxterm expansion of F to maxterm expansion of F'
 - use maxterms whose indices do not appear
 - e.g., $F(A,B,C) = \prod M(0,2,4) \quad F'(A,B,C) = \prod M(1,3,5,6,7)$

Incompletely specified functions

- Example: binary coded decimal increment by 1

- BCD digits encode the decimal digits 0 – 9

- Don't cares and canonical forms

- so far, only represented on-set
- also represent don't-care-set
- need two of the three sets (on-set, off-set, dc-set)

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

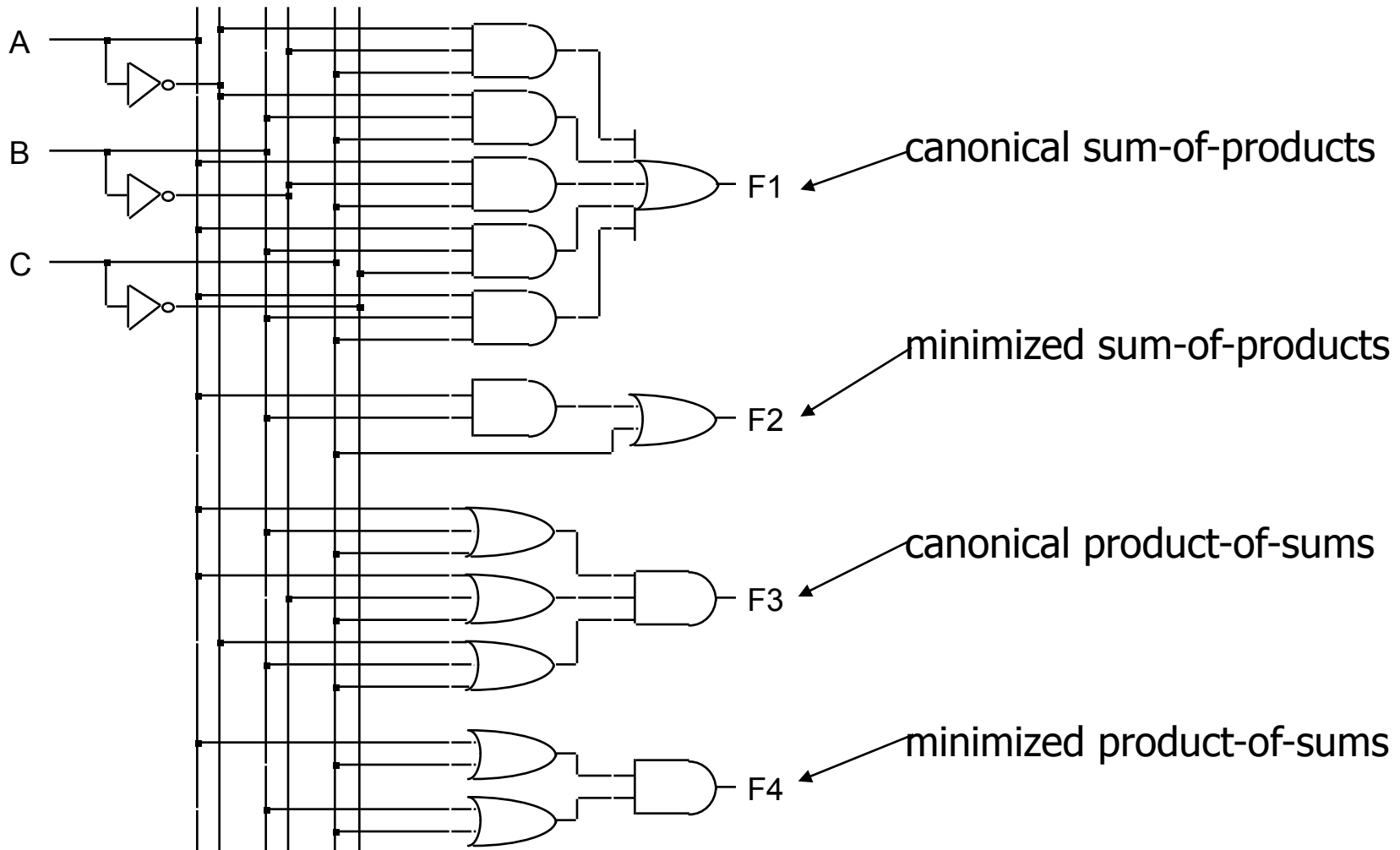
off-set of W

on-set of W

don't care (DC) set of W

these inputs patterns should never be encountered in practice
– **"don't care"** about associated output values, can be exploited in minimization

Alternative two-level implementations of $F = AB + C$



Key to simplification: the uniting theorem

- Uniting theorem: $A(B' + B) = A$
- Essence of simplification of two-level logic
 - find two element subsets of the ON-set where only one variable changes its value – this single varying variable can be eliminated and a single product term used to represent both elements

$$F = A'B' + AB' = (A' + A)B' = B'$$

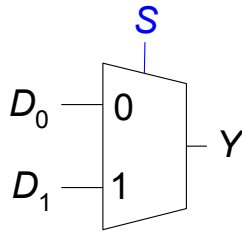
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

B has the same value in both on-set rows
– B remains

A has a different value in the two rows
– A is eliminated

2:1 Multiplexer or Mux

- Selects between one of N inputs to connect to output
- $\log_2 N$ -bit select input – control input
- **Example:** **2:1 Mux**



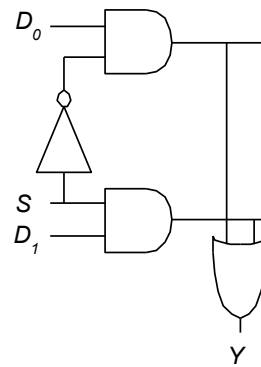
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

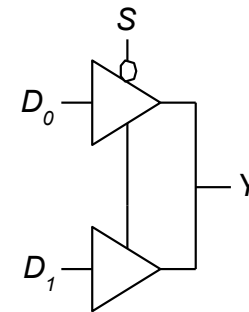
Logic gates

$D_0 D_1$	00	01	11	10
S				
0	0	0	1	1
1	0	1	1	0

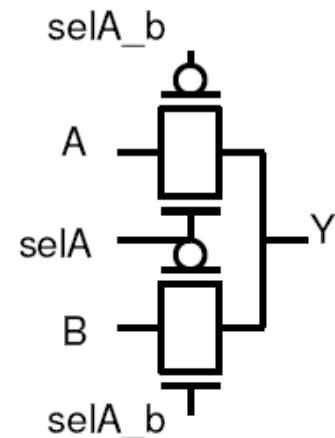
$Y = D_0 \bar{S} + D_1 S$



Tristates



Pass gates

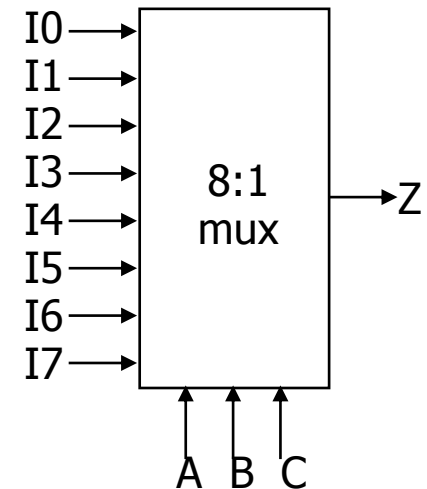
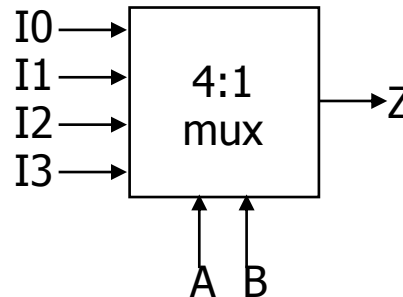
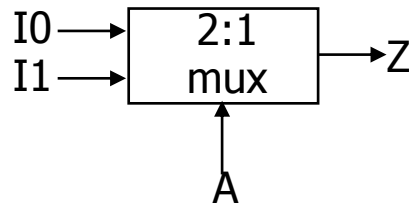


Multiplexers

- 2:1 mux: $Z = A'I_0 + AI_1$
- 4:1 mux: $Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$
- 8:1 mux: $Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7$

- In general: $Z = \sum_{k=0}^{2^n-1} m_k I_k$

– in minterm shorthand form for a $2^n:1$ Mux

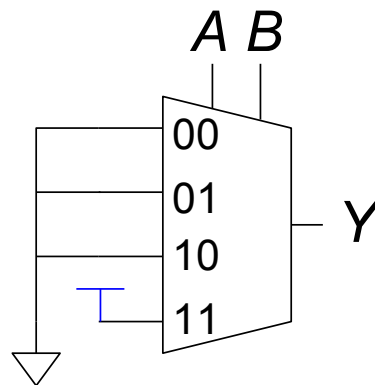


Logic using Multiplexers

- Using the mux as a lookup table

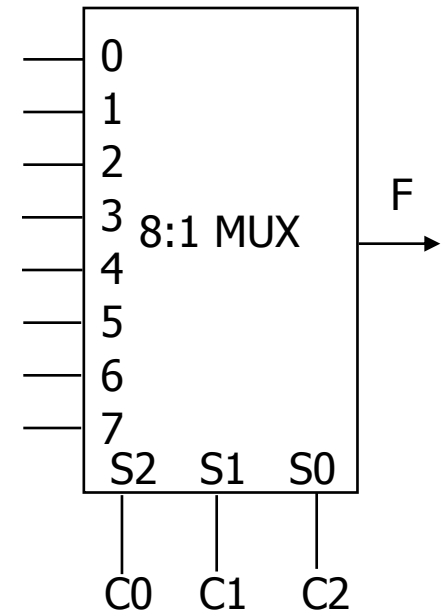
<i>A</i>	<i>B</i>	<i>Y</i>
0	0	0
0	1	0
1	0	0
1	1	1

$$Y = AB$$



Mux example: Logical function unit

C0	C1	C2	Function	Comments
0	0	0	1	always 1
0	0	1	$A + B$	logical OR
0	1	0	$(A \bullet B)'$	logical NAND
0	1	1	$A \text{ xor } B$	logical xor
1	0	0	$A \text{ xnor } B$	logical xnor
1	0	1	$A \bullet B$	logical AND
1	1	0	$(A + B)'$	logical NOR
1	1	1	0	always 0

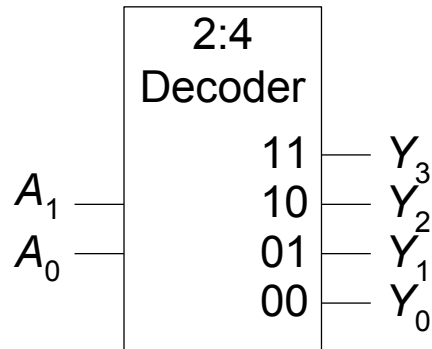


Mux as general-purpose logic

- A $2^{n-1}:1$ multiplexer can implement any function of n variables
 - with $n-1$ variables used as control inputs and
 - the data inputs tied to the last variable or its complement
- Example: $F(A,B,C) = AC + BC' + A'B'C$

Demux or Decoder

- N inputs, 2^N outputs
- One-hot outputs: only one output HIGH at once



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoder: logic equations

- Decoders/demultiplexers
 - control inputs (called “selects” (S)) represent binary index of output to which the input is connected
 - data input usually called “enable” (G)

1:2 Decoder:

$$O0 = G \bullet S'$$

$$O1 = G \bullet S$$

2:4 Decoder:

$$O0 = G \bullet S1' \bullet S0'$$

$$O1 = G \bullet S1' \bullet S0$$

$$O2 = G \bullet S1 \bullet S0'$$

$$O3 = G \bullet S1 \bullet S0$$

3:8 Decoder:

$$O0 = G \bullet S2' \bullet S1' \bullet S0'$$

$$O1 = G \bullet S2' \bullet S1' \bullet S0$$

$$O2 = G \bullet S2' \bullet S1 \bullet S0'$$

$$O3 = G \bullet S2' \bullet S1 \bullet S0$$

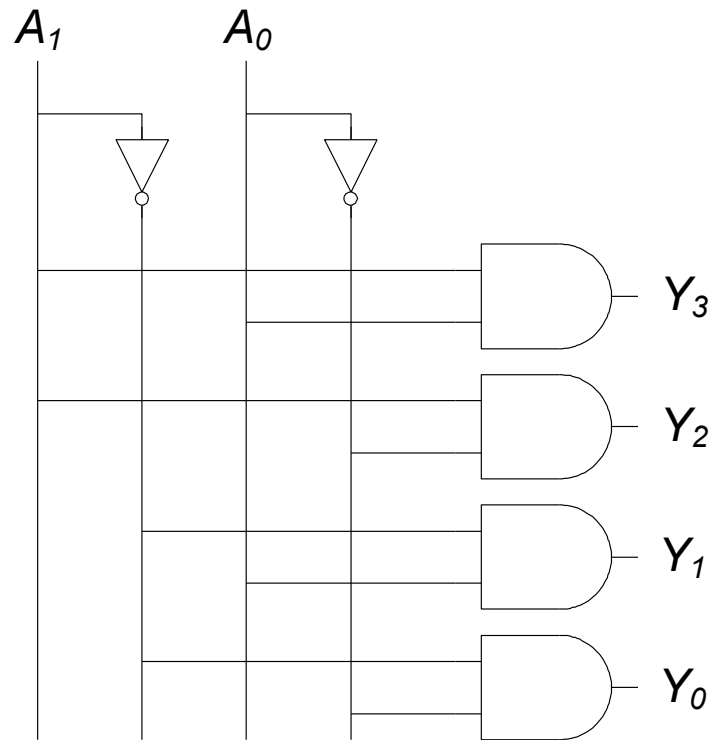
$$O4 = G \bullet S2 \bullet S1' \bullet S0'$$

$$O5 = G \bullet S2 \bullet S1' \bullet S0$$

$$O6 = G \bullet S2 \bullet S1 \bullet S0'$$

$$O7 = G \bullet S2 \bullet S1 \bullet S0$$

Decoder Implementation



Multiple-Output Circuits

- Many circuits have more than one output
- Can give each a separate circuit, or can share gates
- Ex: $F = ab + c'$, $G = ab + bc$

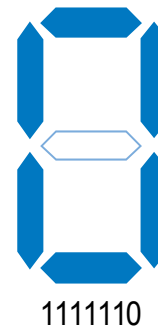
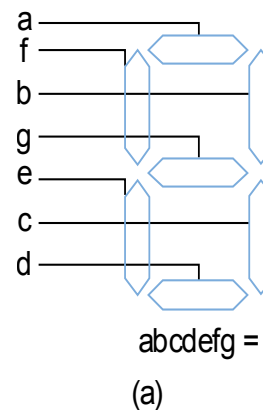
Option 1: Separate circuits

Option 2: Shared gates

Multiple-Output Example: BCD to 7-Segment Converter

TABLE 2-4 4-bit binary number to seven-segment display truth table

w	x	y	z	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

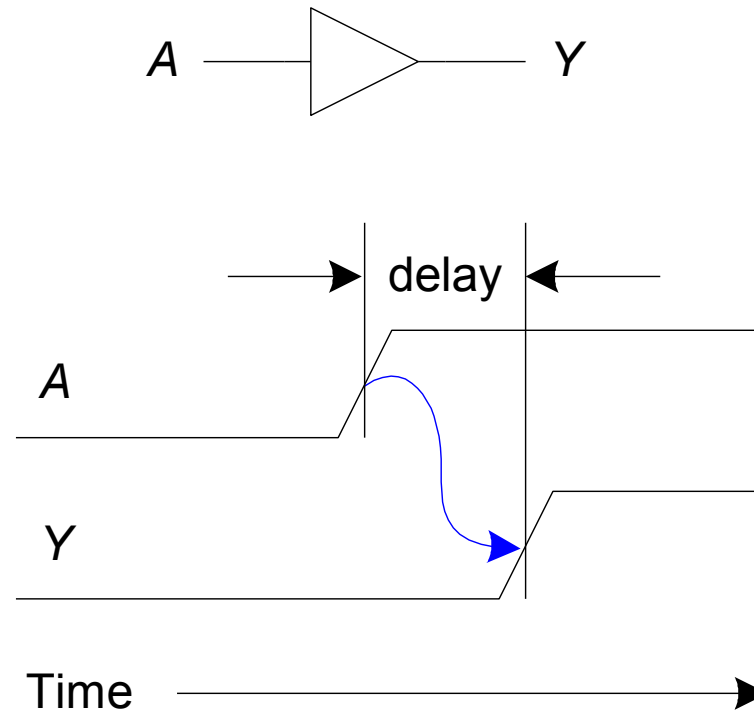


$$a = w'x'y'z' + w'x'yz' + w'x'yz + w'xy'z + w'xyz' + w'xyz + wx'y'z' + wx'y'z$$

$$b = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz + w'xy'z' + w'xyz + wx'y'z' + wx'y'z$$

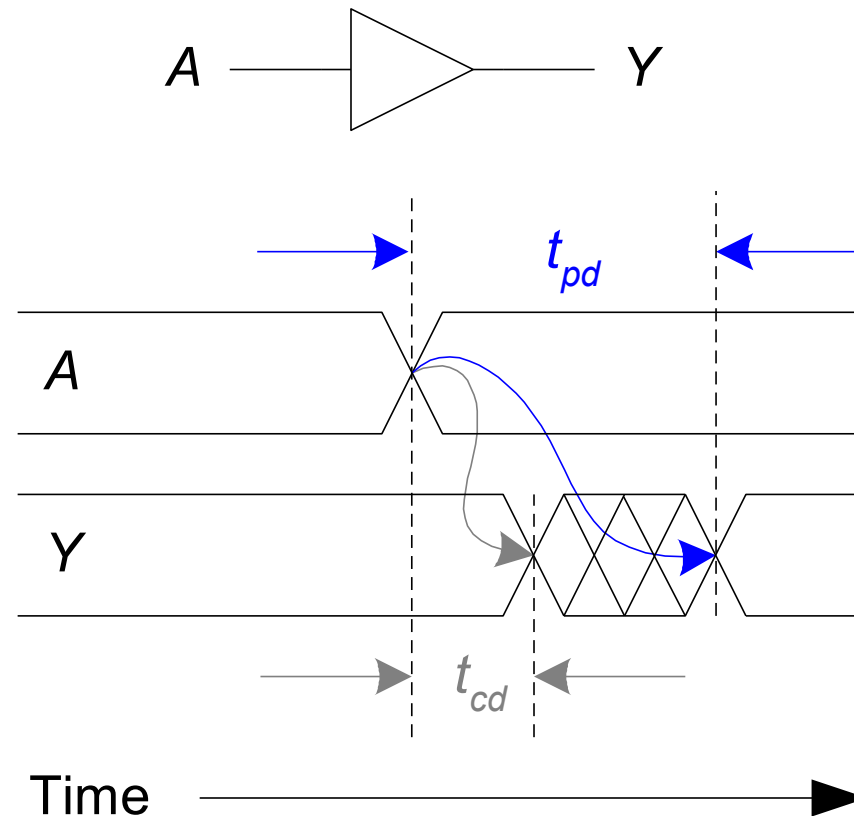
Timing

- Delay between input change and output changing
- How to build fast circuits?



Propagation & Contamination Delay

- **Propagation delay:** $t_{pd} = \text{max delay from input to output}$
- **Contamination delay:** $t_{cd} = \text{min delay from input to output}$

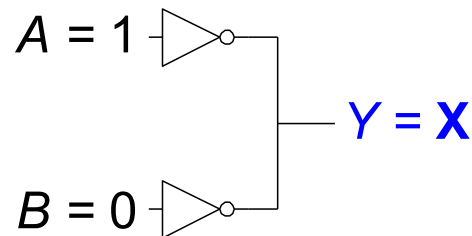


Propagation & Contamination Delay

- Delay is caused by
 - Capacitance and resistance in a circuit
 - Speed of light limitation
- Reasons why t_{pd} and t_{cd} may be different:
 - Different rising and falling delays
 - Multiple inputs and outputs, some of which are faster than others
 - Circuits slow down when hot and speed up when cold

Contention: X

- Contention: circuit tries to drive output to 1 **and** 0
 - Actual value somewhere in between
 - Could be 0, 1, or in forbidden zone
 - Might change with voltage, temperature, time, noise
 - Often causes excessive power dissipation

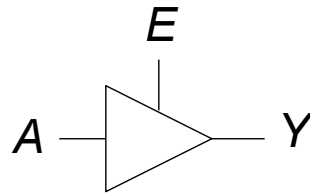


- **Warnings:**
 - Contention usually indicates a **bug**.
 - **X is used for “don’t care” and contention** - look at the context to tell them apart

Floating: Z

- Floating, high impedance, open, high Z
- Floating output might be 0, 1, or somewhere in between
 - A voltmeter won't indicate whether a node is floating

Tristate Buffer



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Tristate Busses

- Floating nodes are used in tristate busses
 - many different drivers, but only one is active at once

