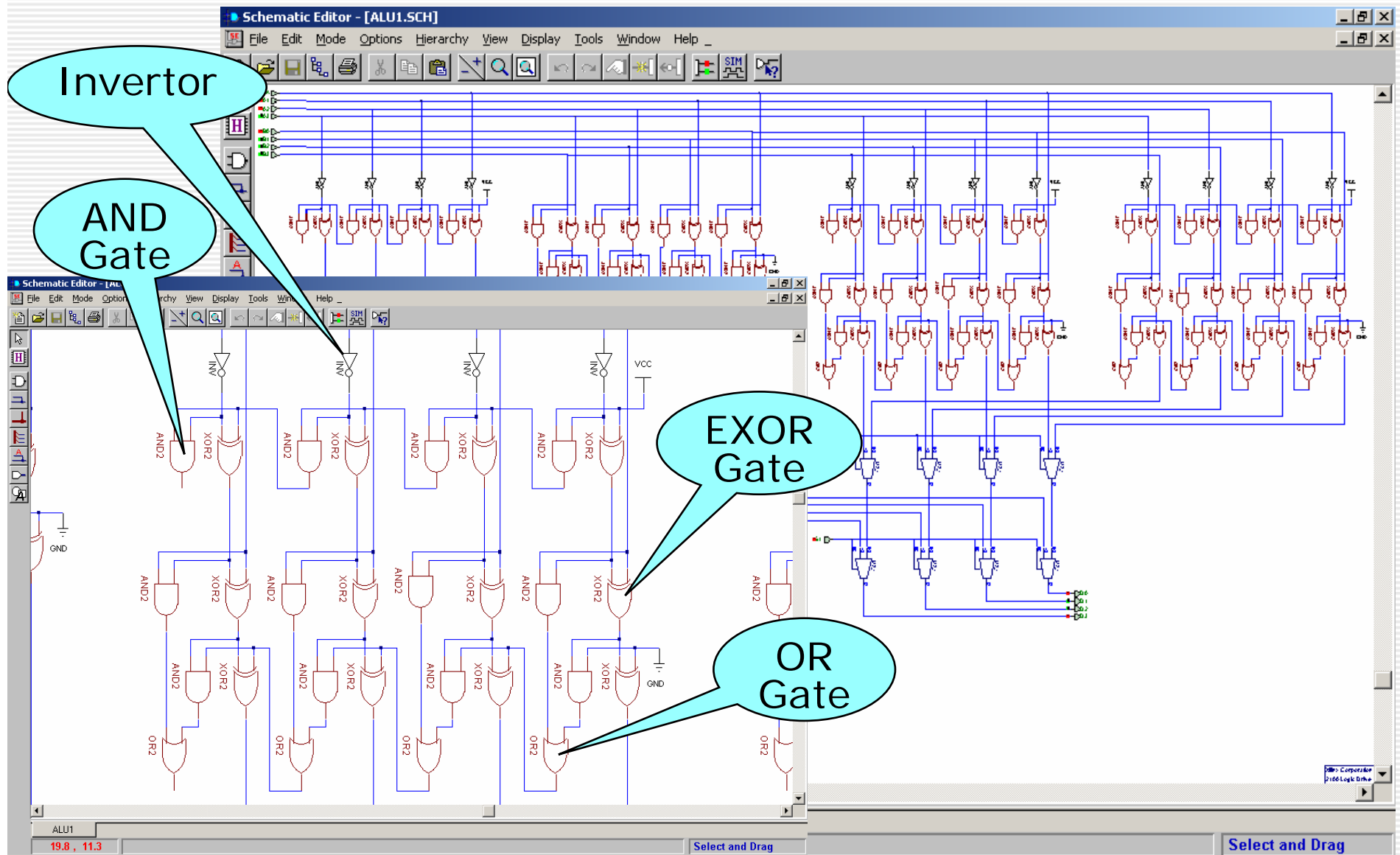


Combinational Logic Circuits

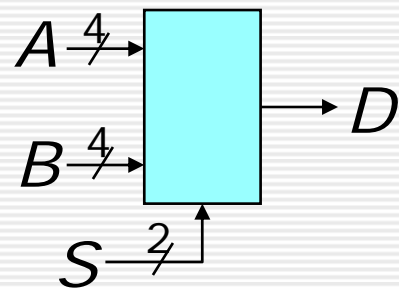
- Binary Logic and Gates (W 79-84)
- Intro to Computer-Aided Design
- Boolean Algebra (W 183-196)
- Standard Forms (W 196-199)
- Circuit Optimization (W 205-230)
- NAND/NOR and XOR gates (W 86-93)
- CMOS and circuit delays (W 97-122)



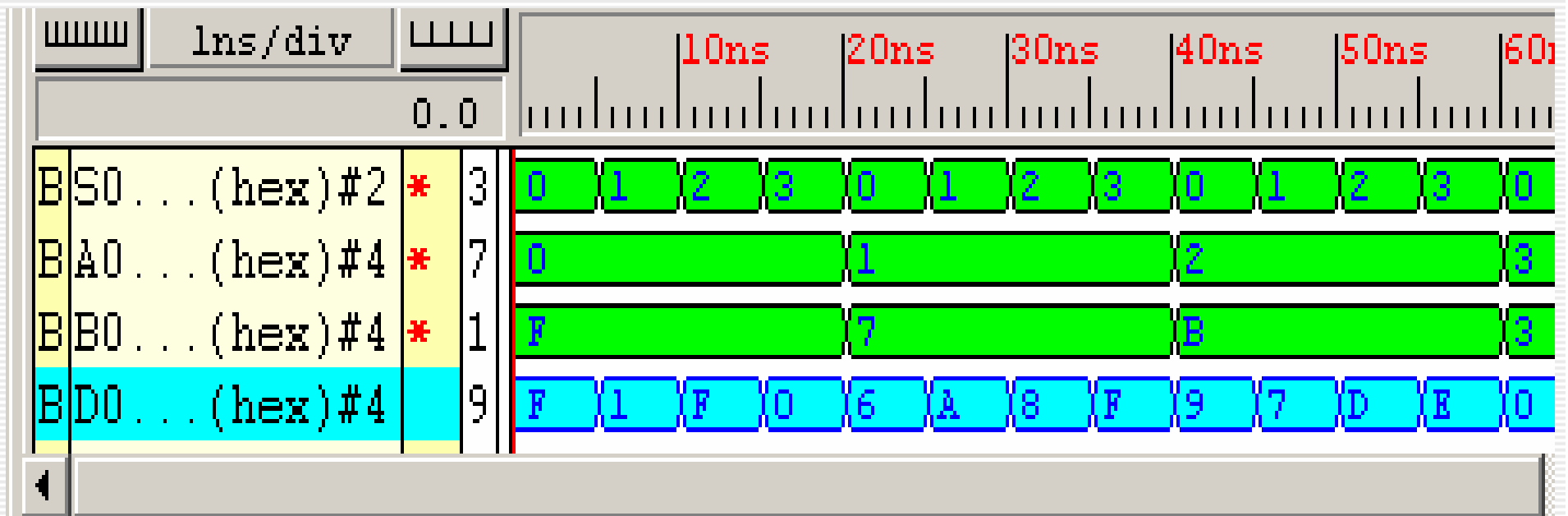
Schematic for 4 Bit ALU



Simulation of 4 Bit ALU



if $S=0$ then $D=B-A$
 if $S=1$ then $D=A-B$
 if $S=2$ then $D=A+B$
 if $S=3$ then $D=-A$



Elementary Binary Logic Functions

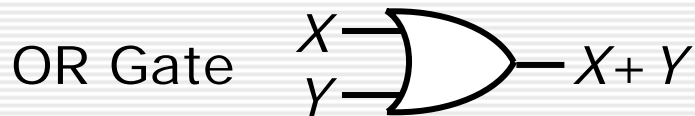
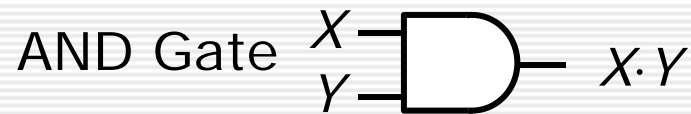
- Digital circuits represent information using two voltage levels.
 - » *binary variables* are used to denote these values
 - » by convention, the values are called "1" and "0" and we often think of them as meaning "True" and "False"
- Functions of binary variables called *logic functions*.
 - » $\text{AND}(A, B) = 1$ if $A=1$ and $B=1$, else it is zero.
 - AND is generally written in shorthand $A \cdot B$ (or $A \& B$ or $A \wedge B$)
 - » $\text{OR}(A, B) = 1$ if $A=1$ or $B=1$, else it is zero.
 - OR is generally written in shorthand form $A + B$ (or $A | B$ or $A \vee B$)
 - » $\text{NOT}(A) = 1$ if $A=0$ else it is zero.
 - NOT is generally written in shorthand form \bar{A} (or $\neg A$ or A')
- AND, OR and NOT can be used to express all other logic functions.

Two Variable Binary Logic Functions

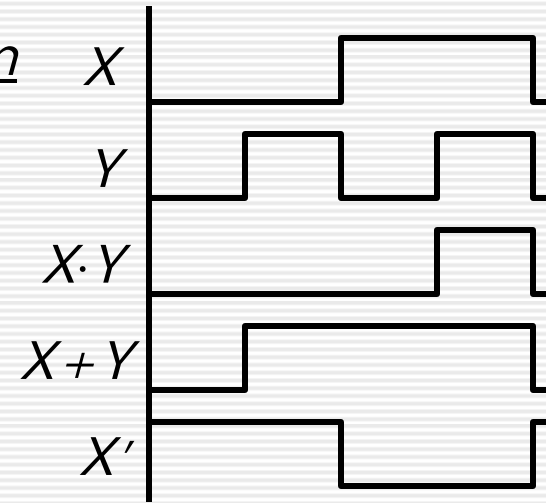
A	B	ZERO	ONE	A	B	A'	B'	AND	NAND	OR	NOR	EXOR	EQUAL	A \Rightarrow B	B \Rightarrow A	(A \Rightarrow B)'	(B \Rightarrow A)'
0	0	0	1	0	0	1	1	0	1	0	1	0	1	1	1	0	0
0	1	0	1	0	1	1	0	0	1	1	0	1	0	1	0	0	1
1	0	0	1	1	0	0	1	0	1	1	0	1	0	0	1	1	0
1	1	0	1	1	1	0	0	1	0	1	0	0	1	1	1	0	0

- Can make similar *truth tables* for 3 variable or 4 variable functions, but gets big (256 & 65,536 cols).
- Representing functions in terms of AND, OR, NOT.
 - » $\text{NAND}(A, B) = (A \cdot B)'$
 - » $\text{EXOR}(A, B) = (A' \cdot B) + (A \cdot B')$

Basic Logic Gates



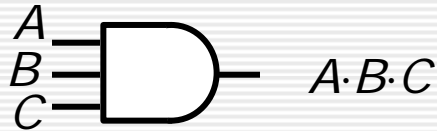
Timing Diagram



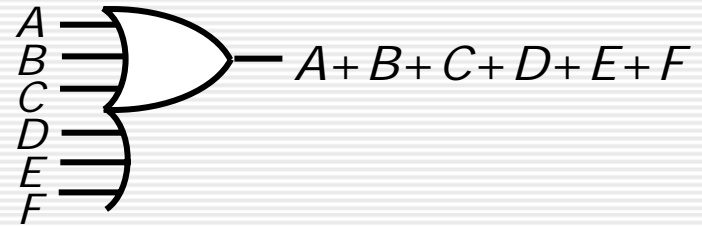
- Logic gates “compute” elementary binary functions.
 - » output of an AND gate is “1” when both of its inputs are “1”, otherwise the output is zero
 - » similarly for OR gate and inverter
- Timing diagram shows how output values change over time as input values change.

Multivariable Gates

3 input AND Gate



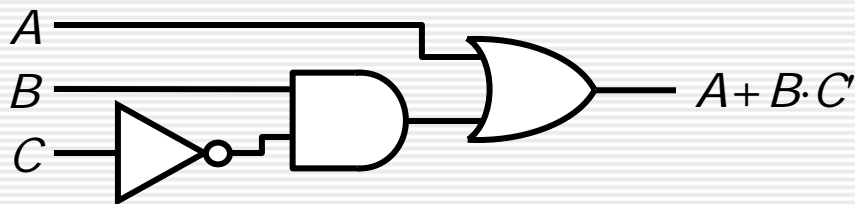
6 input OR Gate



- AND function on n variables is "1" if and only if ALL its arguments are "1".
 - » n input AND gate output is "1" if all inputs are "1"
- OR function on n variables is "1" if and only if at least one of its arguments is "1".
 - » n input OR gate output is "1" if any inputs are "1"
- Can construct "large" gates from 2 input gates.
 - » however, large gates can be less expensive than required number of 2 input gates

Elements of Boolean Algebra

- Boolean algebra defines rules for manipulating symbolic binary logic expressions.
 - » a symbolic binary logic expression consists of binary variables and the operators AND, OR and NOT (e.g. $A+B\cdot C'$)
- The possible values for any Boolean expression can be tabulated in a *truth table*.
- Can define circuit for expression by combining gates.



A	B	C	$B \cdot C'$	$A + B \cdot C'$
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	1
1	1	1	0	1

Schematic Capture & Logic Simulation

The image shows the Xilinx ECS software interface. The main window displays a schematic diagram with various components and connections. The components include gates (AND, OR, NOT), wires, and terminals. The schematic is annotated with callouts: "schematic entry tools" points to the toolbar, "wires" points to the connections, "terminals" points to the input/output points, "gates" points to the logic symbols, and "signal waveforms" points to the waveform viewer. The waveform viewer shows four signals: /testbench/a, /testbench/b, /testbench/c, and /testbench/x. The signals are shown as digital waveforms over time, with a scale of 100 ns. The waveform viewer also shows a time scale of 360000 ps. The symbols list on the left includes: and12, and16, and2, and2b1, and2b2. The Symbol Name Filter is empty. The Orientation is Rotate 0. The status bar shows "Ready" and "ps to 189 ns".

schematic entry tools

wires

terminals

schematic symbols

gates

signal waveforms

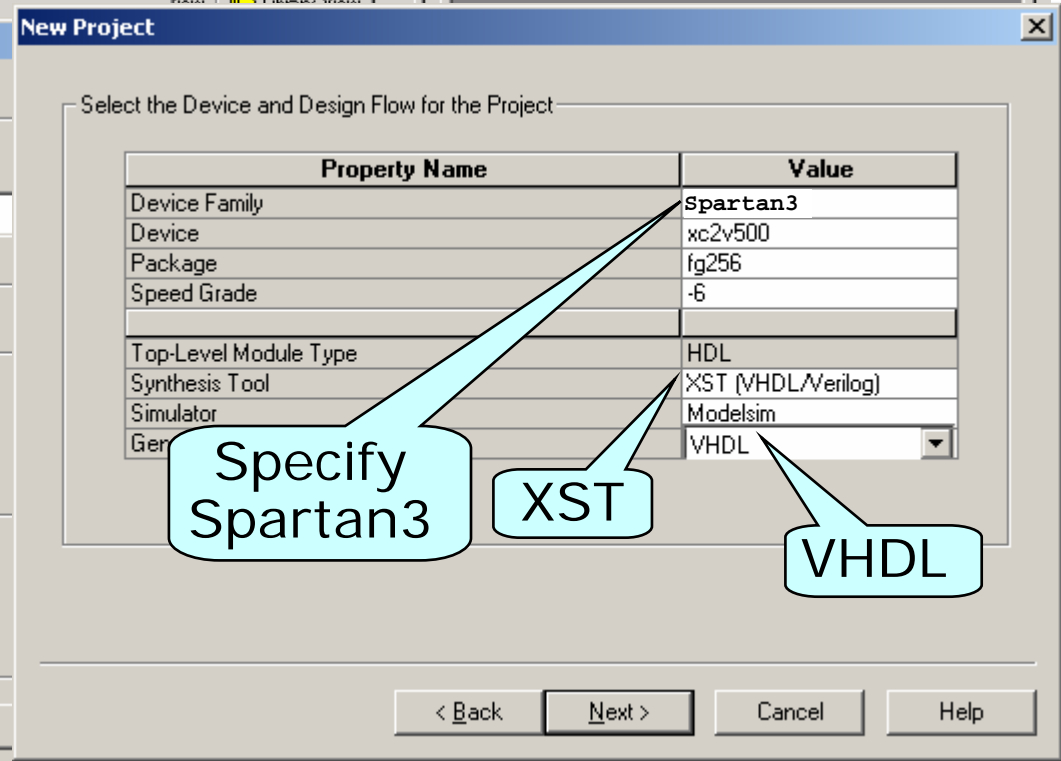
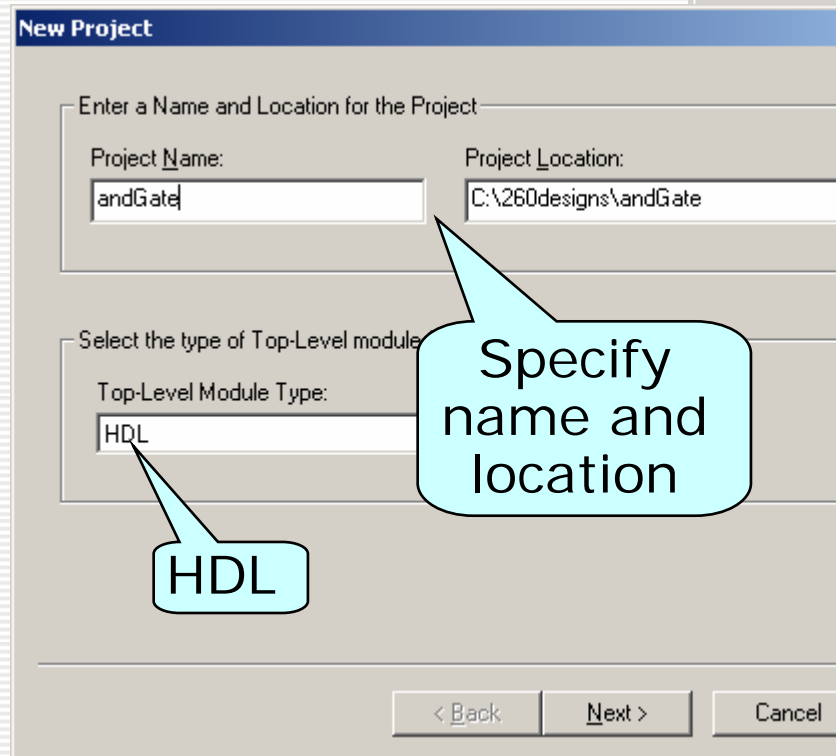
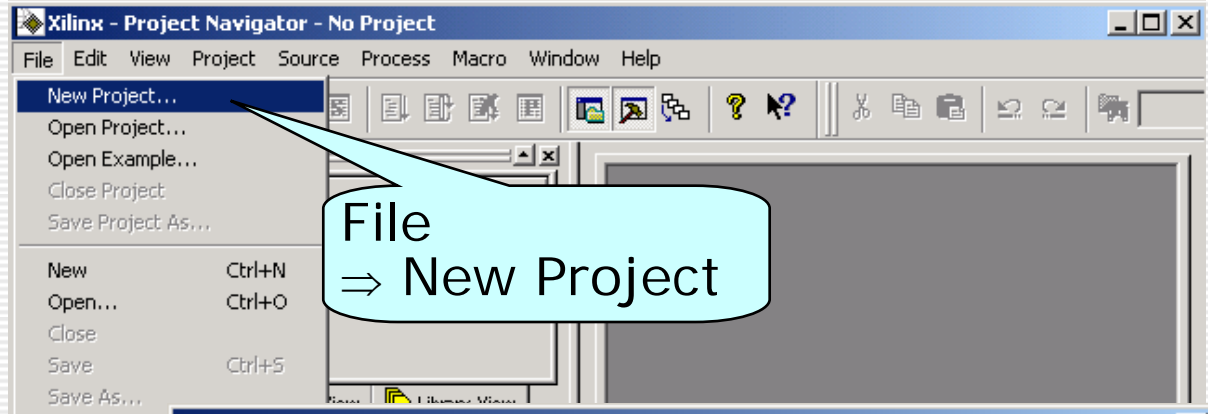
signal names

Outline of Installation

- Register for Xilinx University program (XUP)
 - » <http://xilinx.com/univ/index.htm>
- Insert disk 1 in CD drive and follow on-screen instructions.
 - » registration id on inside cover of CD folder – save this!
 - » install design tools, accepting the defaults if you can
 - **do not install in “Program Files” or any other path with spaces**
- Insert disk 2 in CD drive and follow on-screen instructions
 - » install all the selected tools (including the programming drivers)
 - » if you want to save some disk space, only install the Spartan device files, although installing all devices is fine
- Download other packages from Xilinx – (www.support.xilinx.com)
 - » under “Software Updates – PC” select and download
 - ISE 6.3i Windows - Service Pack – after downloading, execute it
 - ISE 6.3i PC - IP Update – extract using Winzip into directory with Xilinx tools
- Download and install Modelsim – (www.xilinx.com/ise/mxe2/)
 - » *do not purchase*, just download zip file, extract into temp directory and run Setup.exe – during setup specify free starter version
 - » select Full VHDL when prompted for the language support
 - » after installation, select
 - Start ⇒ Programs ⇒ Modelsim ⇒ Submit License Request

Starting New Project

- Start Project Navigator by selecting
 - » Start
 - ⇒ Programs
 - ⇒ Xilinx ISE 6
 - ⇒ Project Navigator



Starting New Project

The screenshot shows the Xilinx Project Navigator interface. The 'Project' menu is open, and the 'New Source...' option is selected. A callout bubble points to this menu item with the text 'Project => New Source'. The 'New Source' dialog box is open, showing a list of source types. The 'Schematic' option is selected, and a callout bubble points to it with the text 'Select Schematic'. The 'File Name' field contains 'andGate', and a callout bubble points to it with the text 'enter name'. The 'Location' field contains 'c:\260designs\andgate'. The 'Add to project' checkbox is checked. The dialog box has buttons for '< Back', 'Next >', 'Cancel', and 'Help'. The status bar at the bottom of the Project Navigator window says 'Add a new source to the project'.

Entering Schematic

The image shows a screenshot of the Xilinx ECS schematic editor interface. The window title is "Xilinx ECS - [andGate.sch]". The menu bar includes File, Edit, View, Add, Tools, Window, and Help. The toolbar contains various icons for file operations, editing, and zooming. On the left, the "Symbols" tab is active, showing a list of categories and a list of symbols. The "and2" symbol is selected. Below the symbol list, there is a "Symbol Name Filter" field and an "Orientation" dropdown set to "Rotate 0". A "Symbol Info" button is at the bottom of the panel. The main workspace is a grid with a blue and2 gate symbol placed on it. Callouts point to the "symbol tab", the "zoom controls" (minus icon), the "select and2" symbol in the list, and the "click here to create gate" symbol on the grid. The status bar at the bottom shows "Ready" and coordinates "[860,991] virtex2".

symbol tab

zoom controls

select and2

click here to create gate

Entering Schematic

The screenshot shows the Xilinx ECS schematic editor interface. The title bar reads "Xilinx ECS - [andGate.sch]". The menu bar includes "File", "Edit", "View", "Add", "Tools", "Window", and "Help". The toolbar contains various icons for file operations, editing, and schematic entry. Below the toolbar are two tabs: "Options" and "Symbols".

The "Options" tab is active, showing the "Add I/O Marker Options" dialog. It contains the following text and options:

When you click near the end of a branch, what do you want to do?

- Add an input marker
- Add an output marker
- Add a bidirectional marker
- Remove the marker

When you add an I/O marker, set its orientation so direction from its connection point is to the --

Automatic

In addition to clicking on a branch end point, you can drag a rectangle around one or more branch end points to add or remove I/O markers at those points

The main schematic area shows a grid with a blue wire connected to an AND gate labeled "AND2". The wire is labeled "XLXN_1". A callout bubble points to the wire with the text "add wires". Another callout bubble points to the AND gate with the text "add IO pins". A third callout bubble points to the "Options" dialog with the text "options for IO tool". A fourth callout bubble points to the "Add I/O Marker Options" dialog with the text "IO pin tool". A fifth callout bubble points to the wiring tool icon in the toolbar with the text "wiring tool".

The status bar at the bottom shows "Ready" on the left, "[1554,838] virtex2" on the right, and "andGate.sch" in the center.

Entering Schematic

The screenshot displays the Xilinx ECS software interface. The main window shows a schematic diagram of an AND gate (AND2) with two inputs labeled XLXN_1 and XLXN_2, and one output labeled XLXN_3. The Object Properties dialog box is open, showing the Net Attributes table. A callout bubble points to the 'Name' field in the table, stating 'rename as desired'. Another callout bubble points to the 'XLXN_3' label in the schematic, stating 'double-click to re-size label'. A third callout bubble points to the output label 'XLXN_3' in the schematic, stating 'double-click for properties'.

Object Properties

Category

- Nets
 - XLXN_3
- I/O Markers
 - XLXN_3

Net Attributes

View and edit the attributes of the selected nets

Name	Value	Visible
Name	XLXN_3	Add
PortPolarity	Output	Add

Buttons: New, Edit Traits, Delete, OK, Cancel, Apply, Help

Preparing to Simulate

The screenshot shows the Xilinx Project Navigator interface. The 'Project' menu is open, and the 'New Source...' option is selected. A callout bubble points to this menu item with the text 'Project => New Source'. Below the menu, the 'New Source' dialog box is open, showing a list of source types. The 'Test Bench Waveform' option is selected, and a callout bubble points to it with the text 'select test bench waveform'. The dialog box also shows the 'File Name' field containing 'testit' and the 'Location' field containing 'c:\260designs\andgate'. The 'Add to project' checkbox is checked. The 'Processes for Current Source' panel shows 'Design Entry Utilities' selected. The 'Console' panel is visible at the bottom.

Project
=> New Source

select test bench waveform

File Name:
testit

Location:
c:\260designs\andgate

Add to project

< Back Next > Cancel Help

Add a new source to the project

Specifying Inputs to Circuit

Initialize Timing

Preview

Assign Inputs → Wait To Check → Check Outputs → Wait To Assign

Clock Timing

Inputs are assigned at 'input setup time' and outputs are checked at 'output valid delay'.

Rising Edge Falling Edge
 Dual Edge (DDR or DET design)

Clock high time: 50 ns
Clock low time: 50 ns
Input setup time: 10 ns
Output valid delay: 10 ns

Design Type

Single Clock
 Multiple Clocks
 Combinatorial Design

Combinatorial Timing

Inputs are assigned, output checked. A delay between assignments avoids assignment/checking conflicts.

Check outputs: 50 ns
Assign inputs: 50 ns

Time Scale: ns

Add Asynchronous Signal Support

Buttons: OK, Next >, Cancel, Help

define input waveforms by clicking to create transitions

note how all combinations of inputs are defined

Comes up on startup - enter 10 in both and ns for Time Scale

Time (ns)	0	20	40	60	80	100
a	0	1	0	1	0	0
b	0	1	0	1	0	0
x	0	1	0	1	0	0

Starting Simulation

The screenshot shows the Xilinx Project Navigator interface. The title bar indicates the project is 'andGate.npl' with a testbench 'testit.tbw'. The menu bar includes File, Edit, View, Project, Source, Process, Options, Window, and Help. The toolbar contains various icons for file operations and simulation. The left pane shows the project hierarchy with 'testit (testit.tbw)' selected. A callout bubble points to this file with the text 'select test file'. The middle pane shows the 'Processes for Source: "testit"' list, with 'Simulate Behavioral Model' highlighted. A callout bubble points to this option with the text 'double-click here to do functional simulation'. The right pane displays a timing diagram for signals 'a', 'b', and 'x' over a 100 ns period. Signal 'a' is high from 20-40 ns and 60-80 ns. Signal 'b' is high from 20-40 ns and 60-80 ns. Signal 'x' is high from 20-40 ns and 60-80 ns. A green text box in the waveform area reads: 'Waveform created by HDL Bench 6.1i, Source = andgate.vhf, Mon Dec 27 11:29:15 2004'. The status bar at the bottom shows 'Process: Simulate Behavioral Model is up to date.'

Running Simulation

The screenshot shows the ModelSim XE II/ Starter 5.8c interface. The main window displays the command window with simulation output, including a PAUSED message. The workspace shows the testbench hierarchy. The waveform window displays the timing diagram for signals /testit/a, /testit/b, and /testit/x. The status bar shows the simulation time from 0 ps to 115500 ps.

main ModelSim command window

select Tools
⇒ Edit Preferences
to change waveform
window colors

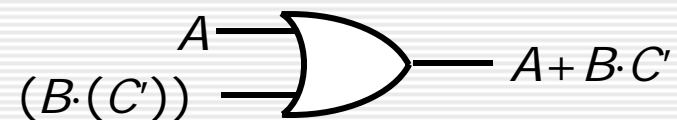
zoom controls

restart and Run -all buttons

waveforms appear here

Boolean Functions to Logic Circuits

- Any Boolean expression can be converted to a logic circuit made up of AND, OR and NOT gates.
 - step 1: add parentheses to expression to fully define order of operations - $A + (B \cdot C')$
 - step 2: create gate for "last" operation in expression
 - gate's output is value of expression
 - gate's inputs are expressions combined by operation



- step 3: repeat for sub-expressions, continue until done
- Number of *simple* gates to implement expression equals number of operations in expression.
 - » simpler *expression* yields less expensive circuit
 - » Boolean algebra provides rules for simplifying logic

Basic Identities of Boolean Algebra

1. $X + 0 = X$	2. $X \cdot 1 = X$	
3. $X + 1 = 1$	4. $X \cdot 0 = 0$	
5. $X + X = X$	6. $X \cdot X = X$	
7. $X + X' = 1$	8. $X \cdot X' = 0$	
9. $(X')' = X$		
10. $X + Y = Y + X$	11. $X \cdot Y = Y \cdot X$	<i>commutative</i>
12. $X + (Y + Z) = (X + Y) + Z$	13. $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$	<i>associative</i>
14. $X(Y + Z) = X \cdot Y + X \cdot Z$	15. $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$	<i>distributive</i>
16. $(X + Y)' = X' \cdot Y'$	17. $(X \cdot Y)' = X' + Y'$	<i>DeMorgan's</i>

- Identities define intrinsic properties of Boolean algebra.
- Note: 15-17 have no counterpart in ordinary algebra.
- Parallel columns illustrate *duality principle*.
- Other handy identities.
 - » $A + AB = A$ (follows from 2, 14 and 3), $A + A'B = A + B$ (15, 7 and 2)

Verifying Identities Using Truth Tables

XY	$(X + Y)' = X' \cdot Y'$	
	$(X + Y)'$	$X' \cdot Y'$
00	1	1
01	0	0
10	0	0
11	0	0

XYZ	$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$				
	$Y \cdot Z$	$X + (Y \cdot Z)$	$X + Y$	$X + Z$	$(X + Y) \cdot (X + Z)$
000	0	0	0	0	0
001	0	0	0	1	0
010	0	0	1	0	0
011	1	1	1	1	1
100	0	1	1	1	1
101	0	1	1	1	1
110	0	1	1	1	1
111	1	1	1	1	1

- Can verify any logical equation with small number of variables using truth tables.
- Break large expressions into parts, as needed.

DeMorgan's Laws for n Variables

- We can extend DeMorgan's laws to 3 variables by applying the laws for two variables.

$$\begin{aligned}(X + Y + Z)' &= (X + (Y + Z))' && \text{- by associative law} \\ &= X' \cdot (Y + Z)' && \text{- by DeMorgan's law} \\ &= X' \cdot (Y' \cdot Z') && \text{- by DeMorgan's law} \\ &= X' \cdot Y' \cdot Z' && \text{- by associative law}\end{aligned}$$

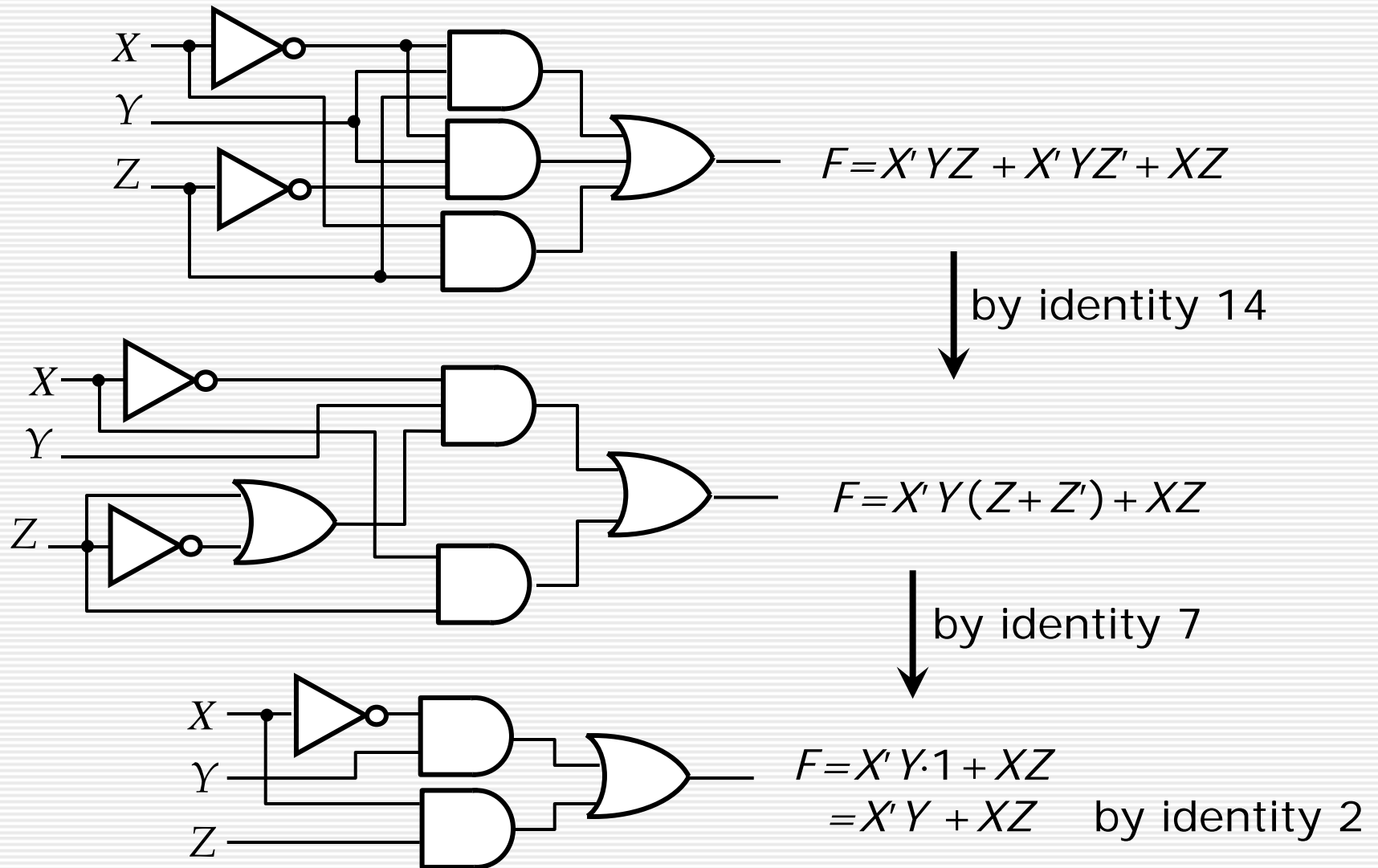
$$\begin{aligned}(X \cdot Y \cdot Z)' &= (X \cdot (Y \cdot Z))' && \text{- by associative law} \\ &= X' + (Y \cdot Z)' && \text{- by DeMorgan's law} \\ &= X' + (Y' + Z') && \text{- by DeMorgan's law} \\ &= X' + Y' + Z' && \text{- by associative law}\end{aligned}$$

- Generalization to n variables.

$$\gg (X_1 + X_2 + \cdots + X_n)' = X'_1 \cdot X'_2 \cdots X'_n$$

$$\gg (X_1 \cdot X_2 \cdots X_n)' = X'_1 + X'_2 + \cdots + X'_n$$

Simplification of Boolean Expressions



The Duality Principle

- The *dual* of a Boolean expression is obtained by interchanging all *ANDs* and *ORs*, and all 0s and 1s.
 - » example: the dual of $A+(B \cdot C')+0$ is $A \cdot (B+C') \cdot 1$
- The duality principle states that if E_1 and E_2 are Boolean expressions then

$$E_1 = E_2 \Leftrightarrow \text{dual}(E_1) = \text{dual}(E_2)$$

where $\text{dual}(E)$ is the dual of E . For example,

$$A+(B \cdot C')+0 = (B' \cdot C)+D \Leftrightarrow A \cdot (B+C') \cdot 1 = (B'+C) \cdot D$$

- » consequently, the pairs of identities (1,2), (3,4), (5,6), (7,8), (10,11), (12,13), (14,15) and (16,17) all follow from each other through the duality principle
- » also, $A+AB=A \Rightarrow A(A+B)=A$
and $A+A'B=A+B \Rightarrow A(A'+B)=AB$

The Consensus Theorem

Theorem. $XY + YZ + X'Z = XY + X'Z$

Proof. $XY + YZ + X'Z$

$$= XY + (X + X')YZ + X'Z \quad 2,7$$

$$= XY + XYZ + X'YZ + X'Z \quad 14$$

$$= XY(1 + Z) + X'Z(Y + 1) \quad 2,11,14$$

$$= XY + X'Z \quad 3,2$$

Example. $(A + B)(A' + C) = AA' + AC + A'B + BC$

$$= AC + A'B + BC$$

$$= AC + A'B$$

Dual. $(X + Y)(Y + Z)(X' + Z) = (X + Y)(X' + Z)$

Taking the Complement of a Function

Method 1. Apply DeMorgan's Theorem repeatedly.

$$\begin{aligned}(X(Y'Z' + YZ))' &= X' + (Y'Z' + YZ)' \\ &= X' + (Y'Z')'(YZ)' \\ &= X' + (Y+Z)(Y'+Z')\end{aligned}$$

Method 2. Complement literals and take dual

$$\begin{aligned}(X(Y'Z' + YZ))' &= \text{dual}(X'(YZ + Y'Z')) \\ &= X' + (Y+Z)(Y'+Z')\end{aligned}$$

Sum of Products Form

- The *sum of products* is one of two *standard forms* for Boolean expressions.

$$\langle \text{sum-of-products-expression} \rangle = \langle \text{p-term} \rangle + \langle \text{p-term} \rangle \dots + \langle \text{p-term} \rangle$$
$$\langle \text{p-term} \rangle = \langle \text{literal} \rangle \cdot \langle \text{literal} \rangle \cdot \dots \cdot \langle \text{literal} \rangle$$

» *example.* $X'Y'Z + X'Z + XY + XYZ$

- A *minterm* is a term that contains every variable, in either complemented or uncomplemented form.

» *example.* in expr above, $X'Y'Z$ is minterm, but $X'Z$ is not

- A *sum of minterms expression* is a sum of products expression in which every term is a minterm.

» *example:* $X'Y'Z + X'YZ + XYZ' + XYZ$ is sum of minterms expression that is equivalent to expression above.

» *shorthand:* list minterms numerically, so $X'Y'Z + X'YZ + XYZ' + XYZ$ becomes $001 + 011 + 110 + 111$ or $\Sigma_m(1, 3, 6, 7)$

Simplifying Sum-of-Products Expressions

- Sum of products forms yield 2 level AND-OR circuits.
- Any expression can be put into sum of products form by applying distributive laws.
- The simplest sum of products expression yields simplest 2 level AND-OR circuit.
- Any Boolean expression can be viewed as a *set of minterms*.
- An expression F covers another expression G , if the minterms in G are a subset of the minterms in F .
 - » AC covers $AB'C$, since AC contains minterms 5 and 7 (from the set of 8 minterms on the variables A , B , and C) and $AB'C$ contains only minterm 5.

General Simplification Procedure

Given an expression F (e.g. $ABD + A'B + BC'D' + B'CD + B'CD'$).

Step 1. Let M be the set of minterms covered by F .

$$\begin{array}{cccc} & & A'B'CD & A'B'CD' \\ A'BC'D' & A'BC'D & A'BCD & A'BCD' \\ ABC'D' & ABC'D & ABCD & \\ & & AB'CD & AB'CD' \end{array}$$

Step 2. For each minterm, m , find all *maximal* terms that cover m and also cover other minterms in M , but no minterms that are not in M . Let T be the resulting set of terms. ($T = \{A'B, BC', BD, CD, A'C, B'C\}$)

Step 3. Select all terms in T that cover minterms covered by no other terms in T . ($\{BC', B'C\}$)

Step 4. Select additional terms in T until selected terms cover all minterms. At each step, select a term that covers the largest possible number of new minterms. ($\{A'B, CD\}$)

Simplification Using Karnaugh Maps

Step 1. List all minterms covered by F . Step 3. Select essential terms.

		CD			
		00	01	11	10
AB	00	0	0	1	1
	01	1	1	1	1
	11	1	1	1	0
	10	0	0	1	1

		CD			
		00	01	11	10
AB	00	0	0	1	1
	01	1	1	1	1
	11	1	1	1	0
	10	0	0	1	1

Step 2. Find *maximal* terms.

Step 4. Cover remaining minterms.

		CD			
		00	01	11	10
AB	00	0	0	1	1
	01	1	1	1	1
	11	1	1	1	0
	10	0	0	1	1

		CD			
		00	01	11	10
AB	00	0	0	1	1
	01	1	1	1	1
	11	1	1	1	0
	10	0	0	1	1

More Karnaugh Maps

		BC			
		00	01	11	10
A	0	0	1	0	1
	1	1	1	1	1

$$F = AB'C' + B'C + ABC + BC'$$

$$F = A + B'C + BC'$$

		CD			
		00	01	11	10
AB	00	0	0	0	1
	01	1	1	0	1
	11	1	1	1	1
	10	1	0	1	1

$$F = A'BC' + A'CD' + ABC$$

$$+ AB'C'D' + ABC' + AB'C$$

$$F = BC' + CD' + AC + AD'$$

- Covering 0s gives complement of function.

		CD			
		00	01	11	10
AB	00	0	0	0	1
	01	1	1	0	1
	11	1	1	1	1
	10	1	0	1	1

$$F' = A'B'C' + B'C'D + A'CD$$

If we then take the complement of this expression, we get the product of sums form.

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

Don't Care Conditions

- In some situations, we don't care about the value of a function for certain combinations of the variables.
 - » these combinations may be impossible in certain contexts
 - » or the value of the function may not matter in when the combinations occur
- In such situations we say the function is *incompletely specified* and there are multiple (completely specified) logic functions that can be used in the design.
 - » so we can select a function that gives the simplest circuit
- When constructing the terms of T in the simplification procedure, we can choose to either cover or not cover the don't care conditions.

Map Simplification with Don't Cares

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00	0	1	0	0
	01	x	x	x	1
	11	1	1	1	x
	10	x	0	1	1

$$F = A' C' D + B + AC$$

■ Alternative covering.

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00	0	1	0	0
	01	x	x	x	1
	11	1	1	1	x
	10	x	0	1	1

$$F = A' B' C' D + ABC' + BC + AC$$

Product of Sums Form

- The *product of sums* is the second *standard form* for Boolean expressions.

$$\langle \text{product-of-sums-expression} \rangle = \langle \text{s-term} \rangle \cdot \langle \text{s-term} \rangle \dots \cdot \langle \text{s-term} \rangle$$
$$\langle \text{s-term} \rangle = \langle \text{literal} \rangle + \langle \text{literal} \rangle + \dots + \langle \text{literal} \rangle$$

» *example.* $(X' + Y' + Z)(X' + Z)(X + Y)(X + Y + Z)$

- A *maxterm* is a sum term that contains every variable, in complemented or uncomplemented form.

» *example.* in exp. above, $X' + Y' + Z$ is a maxterm, but $X' + Z$ is not

- A *product of maxterms expression* is a product of sums expression in which every term is a maxterm.

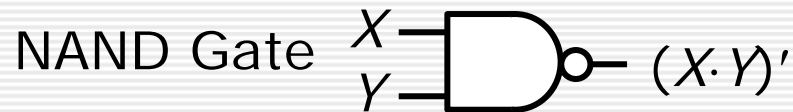
» *example.* $(X' + Y' + Z)(X' + Y + Z)(X + Y + Z')(X + Y + Z)$ is product of maxterms expression that is equivalent to expression above.

» *shorthand:* list maxterms numerically:

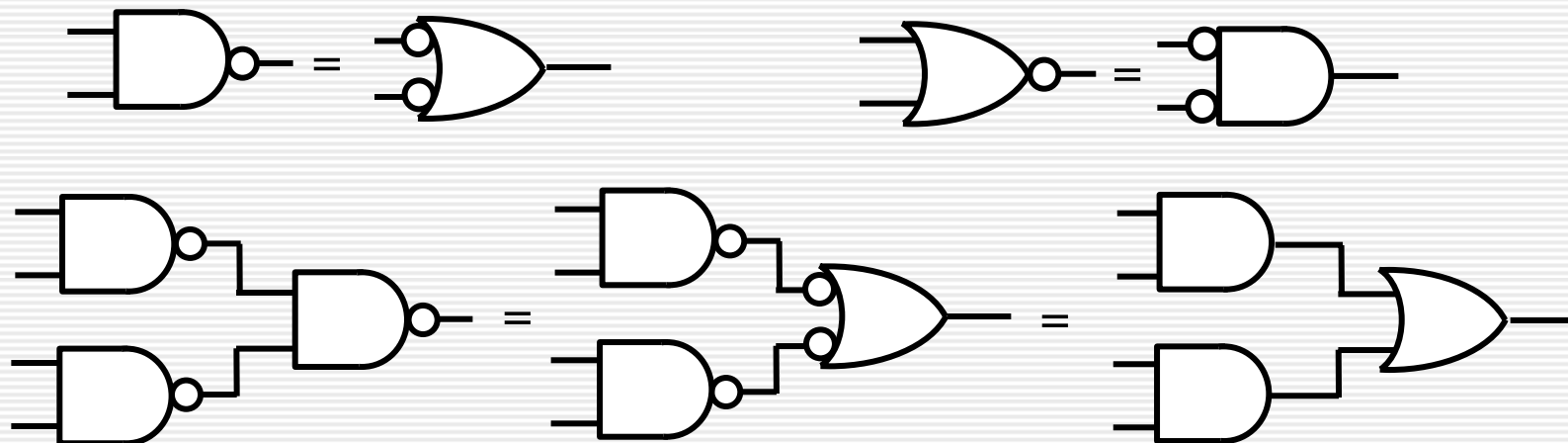
$$\text{so, } (X' + Y' + Z)(X' + Y + Z)(X + Y + Z')(X + Y + Z)$$

becomes $110 + 100 + 001 + 000$ or $\Pi_M(6, 4, 1, 0)$

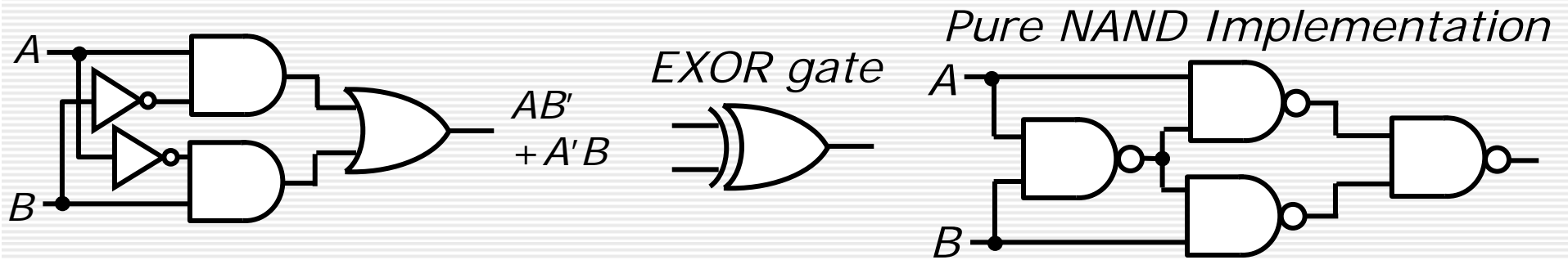
NAND and NOR Gates



- In certain technologies (including CMOS), a NAND (NOR) gate is simpler & faster than an AND (OR) gate.
- Consequently circuits are often constructed using NANDs and NORs directly, instead of ANDs and ORs.
- Alternative gate representations makes this easier.



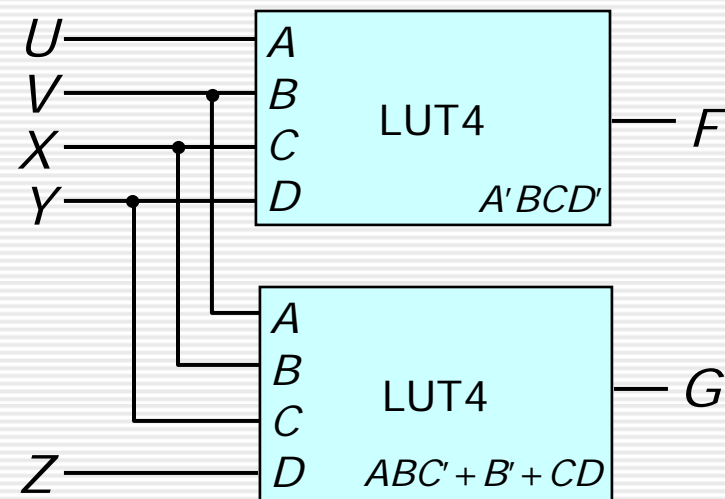
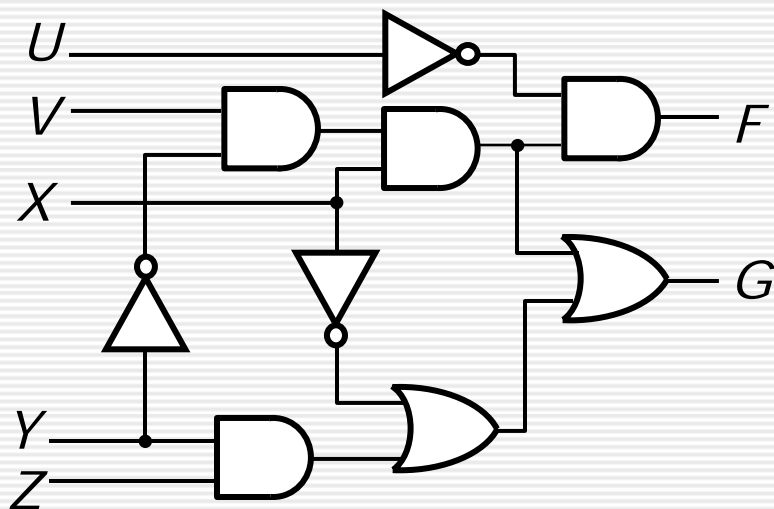
Exclusive Or and Odd Function



- The EXOR function is defined by $A \oplus B = AB' + A'B$.
- The *odd* function on n variables is 1 when an odd number of its variables are 1.
 - » $\text{odd}(X, Y, Z) = XY'Z' + X'YZ' + X'Y'Z + XYZ = X \oplus Y \oplus Z$
 - » similarly for 4 or more variables
- *Parity checking* circuits use the odd function to provide a simple integrity check to verify correctness of data.
 - » any erroneous single bit change will alter value of odd function, allowing detection of the change

Lookup Tables

- Digital logic is often implemented using devices called *Field Programmable Gate Arrays (FPGA)*.
 - » configurable device that can implement many different circuits
 - » implemented using “programmable” logic components and wires
- A *Lookup Table (LUT)* can be configured for any logic function with specified number of inputs (typically 4).
 - » can view LUT as hardware implementation of truth table
- Example of circuit implemented with LUTs:



Integrated Circuits

- Digital logic is implemented using *transistors* in *integrated circuits* containing many gates.
 - » small-scale integrated circuits (SSI) contain 10 gates or less
 - » medium-scale integrated circuits (MSI) contain 10-100 gates
 - » large-scale integrated circuits (LSI) contain up to 10^4 gates
 - » very large-scale integrated circuits (VLSI) contain $>10^4$ gates
- Improvements in manufacturing lead to ever smaller transistors allowing more per chip.
 - » $>10^7$ gates/chip now possible; doubles every 18-24 months
- Variety of logic families.
 - » CMOS - complementary metal-oxide semiconductor
 - » TTL - transistor-transistor logic
 - » ECL - emitter-coupled logic
 - » GaAs - gallium arsenide

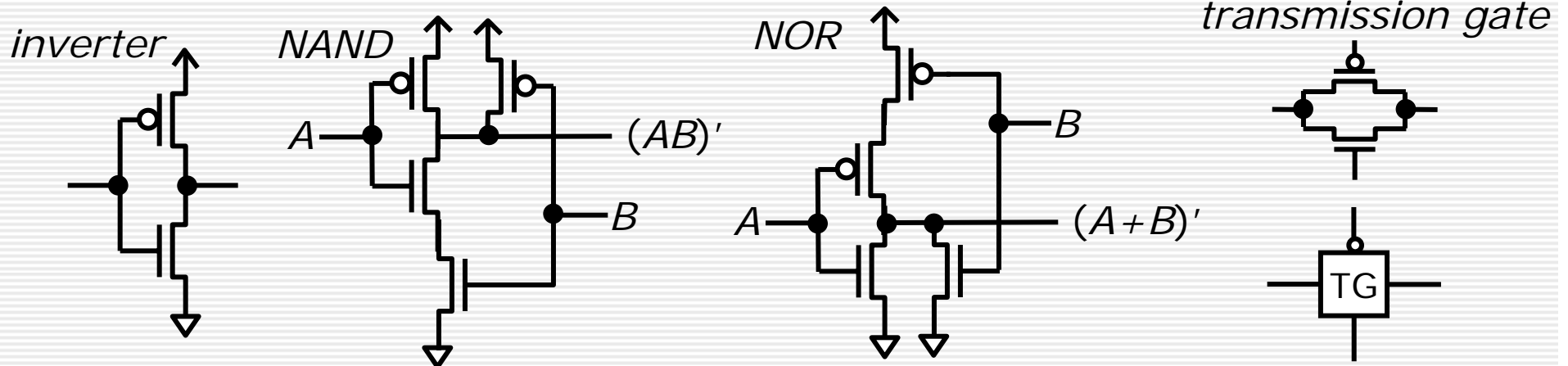
CMOS Logic Gates

- CMOS integrated circuits are built using two types of *Field Effect Transistors* (FET), *n*-type & *p*-type.



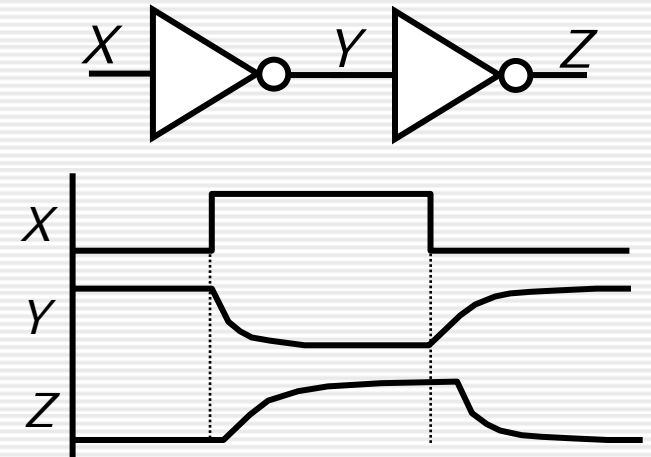
» the *gate* (note different meaning) input controls whether current can flow between the other two terminals or not.

- Logic gates are constructed by combining transistors in complementary arrangements.

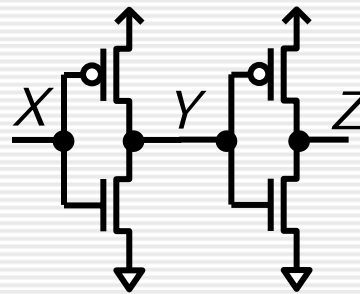
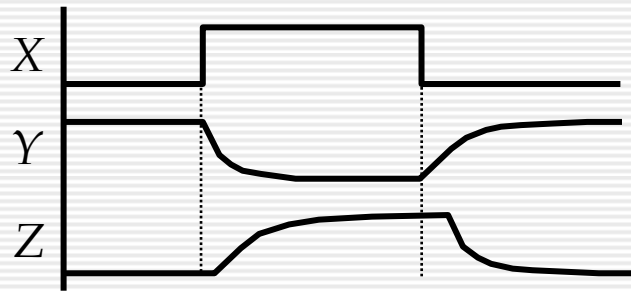


Circuit Delays in CMOS Circuits

- Electronic gates are physical devices that take time to operate.
- Response to instantaneous change at X is gradual decrease in voltage at Y and similar gradual increase at Z .
- Voltage at Y must drop below logic threshold level to be "seen" as a '0'.
- This effect can be viewed as delay in propagation of logic values.
 - » t_{PLH} denotes low-to-high delay
 - » t_{PHL} denotes high-to-low delay
 - » $t_{pd} = \max\{t_{PLH}, t_{PHL}\}$
 - » relative values of t_{PLH} and t_{PHL} depend on relative "strength" of pull-up and pull-down transistors in inverters
 - » values vary with operating temperature and manufacturing processes

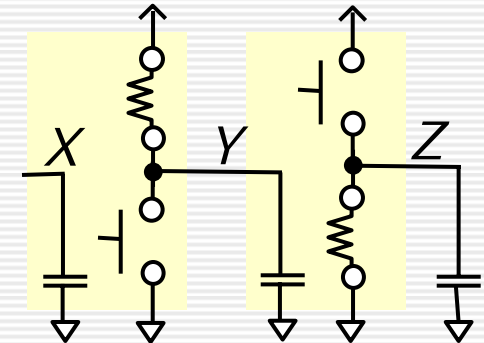


Closer Look at CMOS Circuit Delays

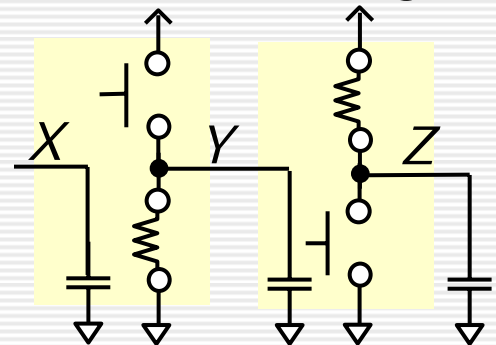


- When X goes high, pull-up of first inverter turns off and pull-down turns on.
- Decrease of voltage at Y requires transfer of charge from capacitor to ground.
 - » wires, transistor gates act like capacitors
 - » time for transfer depends on size of capacitance and on resistance of pull-down transistor
 - » pull-up & pull-down transistors can have different "on-state" resistance values
- Use of two parallel inverters between X and Y can give faster logic transitions.

equivalent circuit when X is low



equivalent circuit when X is high



Negative Logic – What's in a Name?

- In *positive logic* systems, a high voltage is associated with a logic 1, and a low voltage with a logic 0.
 - » positive logic is just one of two *conventions* that can be used to associate a logic value with a voltage
 - » sometimes it is more convenient to use opposite convention
- Circuits often have some signals that are “active low”.
 - » a signal called “enable” may allow some operation to occur only when it is low
 - » it's good practice to label such signals explicitly to prevent confusion - e.g. enable.L
 - » the name of a signal may determine if it's viewed as active high or active low (for example enable.L=inhibit.H)
- To avoid ambiguity, manufacturers generally specify components in terms of high and low voltage values.