Multi-level Logic: Combinatorial Circuits

Becker/Molitor, Chapter 8.1

Jan Reineke
Universität des Saarlandes

Implementation of Boolean functions

Wanted:

- Cheaper representations that need not be based on Boolean polynomials
  - There are Boolean functions whose best representations via Boolean polynomials are very expensive…
- Practical implementation of these representations

Approach:

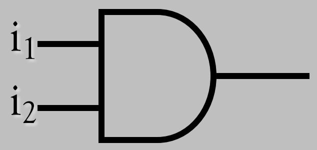
- Find implementations for **simple** Boolean functions
- Compose these to implement more complex functions
  → leads to **hierarchical models**

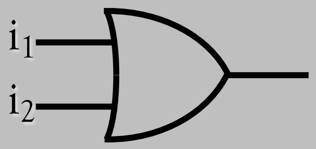
Examples of simple Boolean functions...

i_2	i_1	AND_2
0	0	0
0	1	0
1	0	0
1	1	1

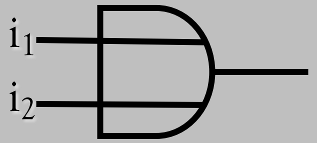
i_2	i_1	OR_2
0	0	0
0	1	1
1	0	1
1	1	1

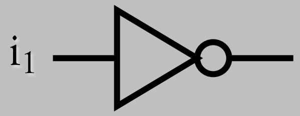
i	NOT
0	1
1	0





also:

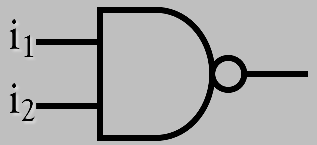


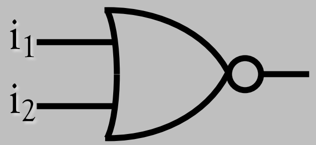


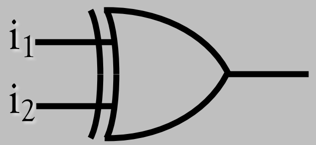
i_2	i_1	$NAND_2$
0	0	1
0	1	1
1	0	1
1	1	0

i_2	i_1	NOR_2
0	0	1
0	1	0
1	0	0
1	1	0

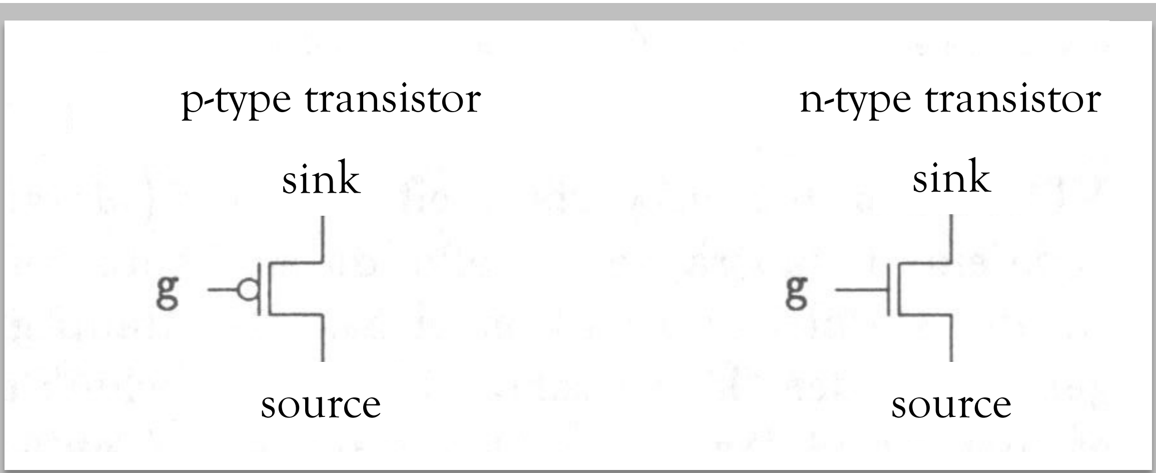
i_2	i_1	XOR_2
0	0	0
0	1	1
1	0	1
1	1	0







Short excursion: Transistors

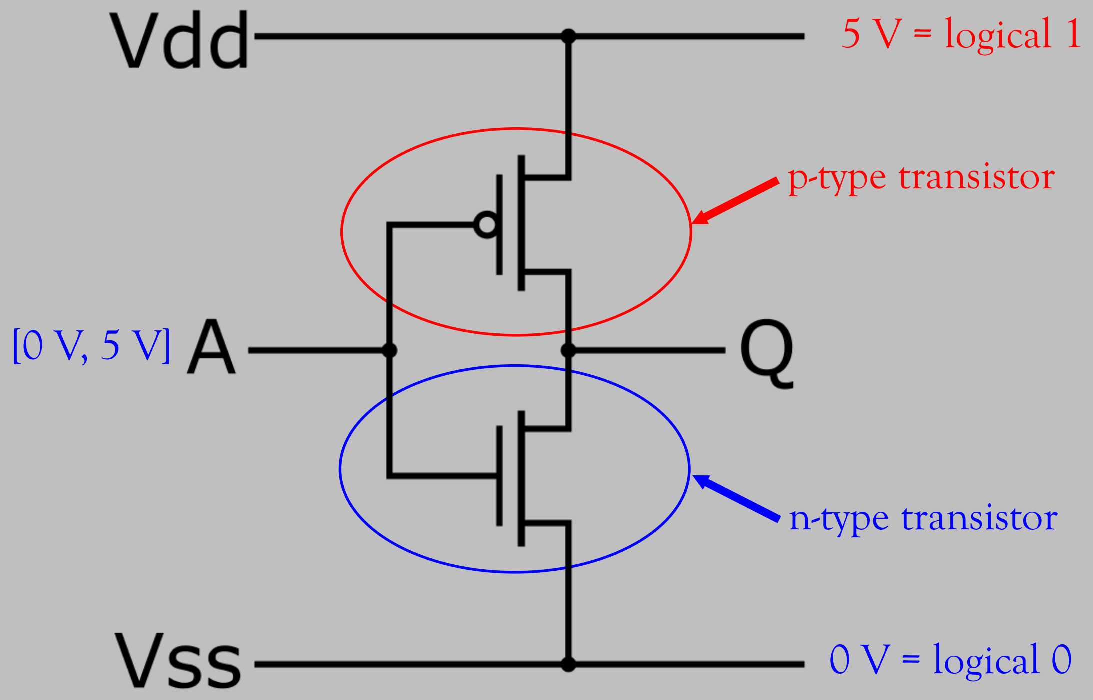


- A transistor can be seen as a voltage-controlled switch:
  - Gate g controls the conductivity between source and sink
- **n-type transistor**:
  - transmits, if gate is 1
  - disconnects, if gate is 0
- **p-type transistor**:
  - transmits, if gate is 0
  - disconnects, if gate is 1

Short excursion: MOS transistors

- CMOS = **C**omplementary **M**etal **O**xide **S**emiconductor
- CMOS uses n-type as well as „complementary“ p-type transistors

Short excursion: CMOS inverter (1/3)



5 V = logical 1

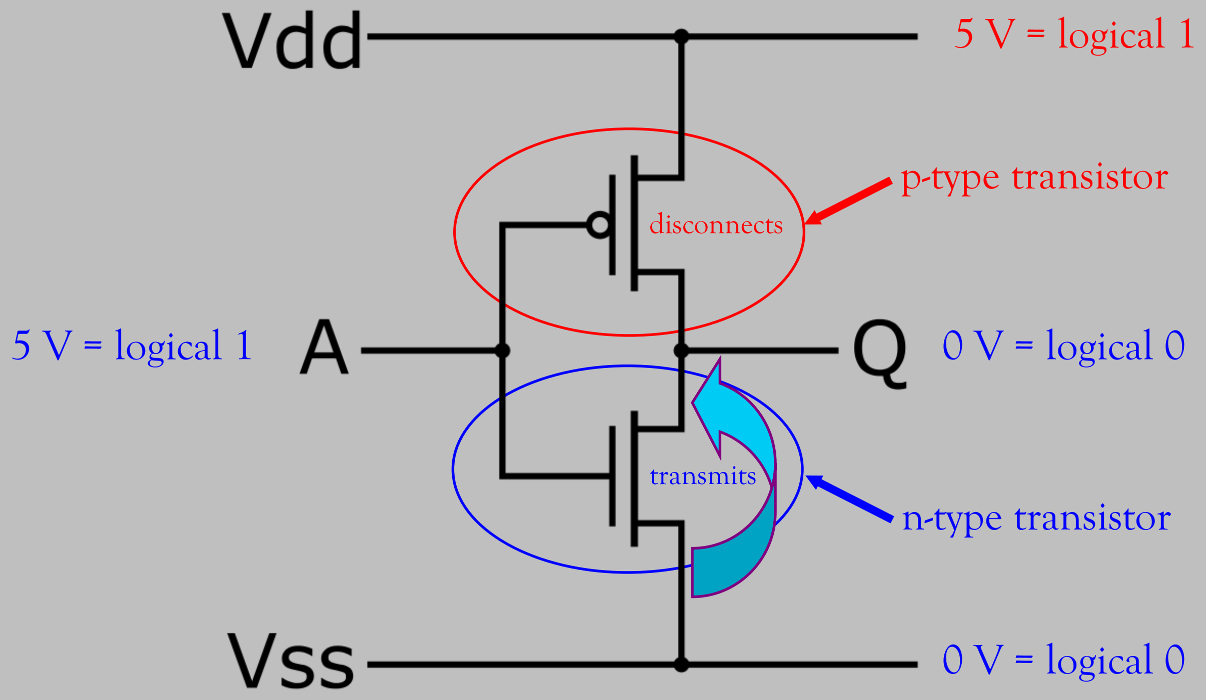
p-type transistor

[0 V, 5 V]

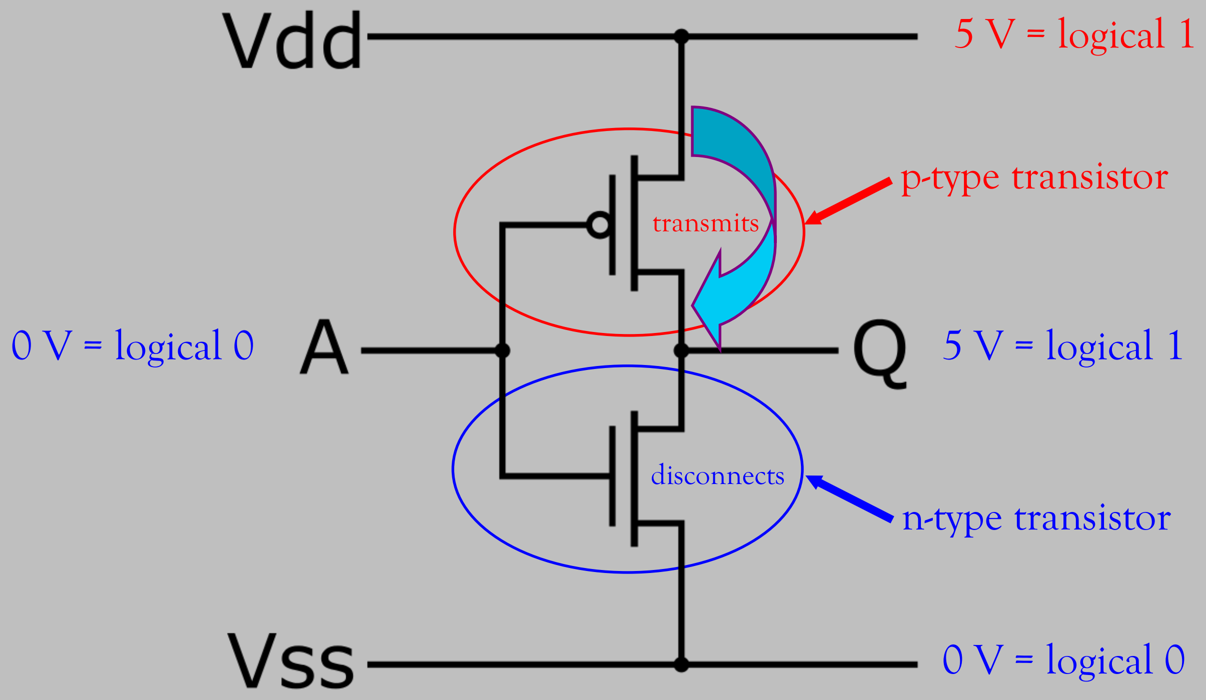
n-type transistor

0 V = logical 0

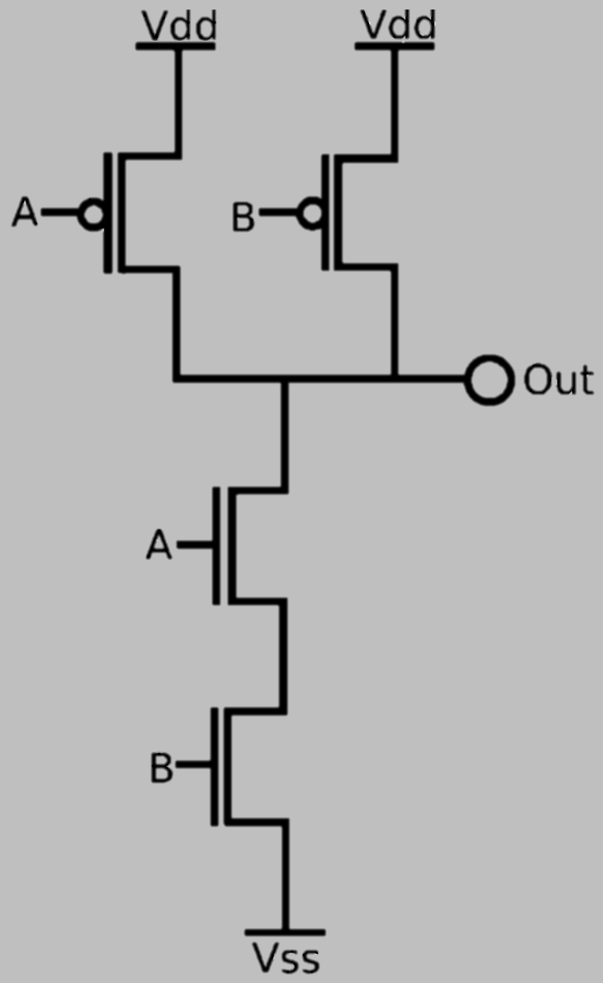
Short excursion: CMOS inverter (2/3)



Short excursion: CMOS inverter (3/3)



Short excursion: CMOS NAND



Output is 0 *iff*
there is a transmitting path from 0 to the output,
i.e., *iff* **both** n-type transistors transmit,
a = b = 1,
then *NAND*(a, b) = 0

Output is 1 *iff*
there is a transmitting path from 1 to the output,
i.e., *iff* **one** of the p-type transistors transmits,
a = 0 or b = 0,
then *NAND*(a, b) = 1

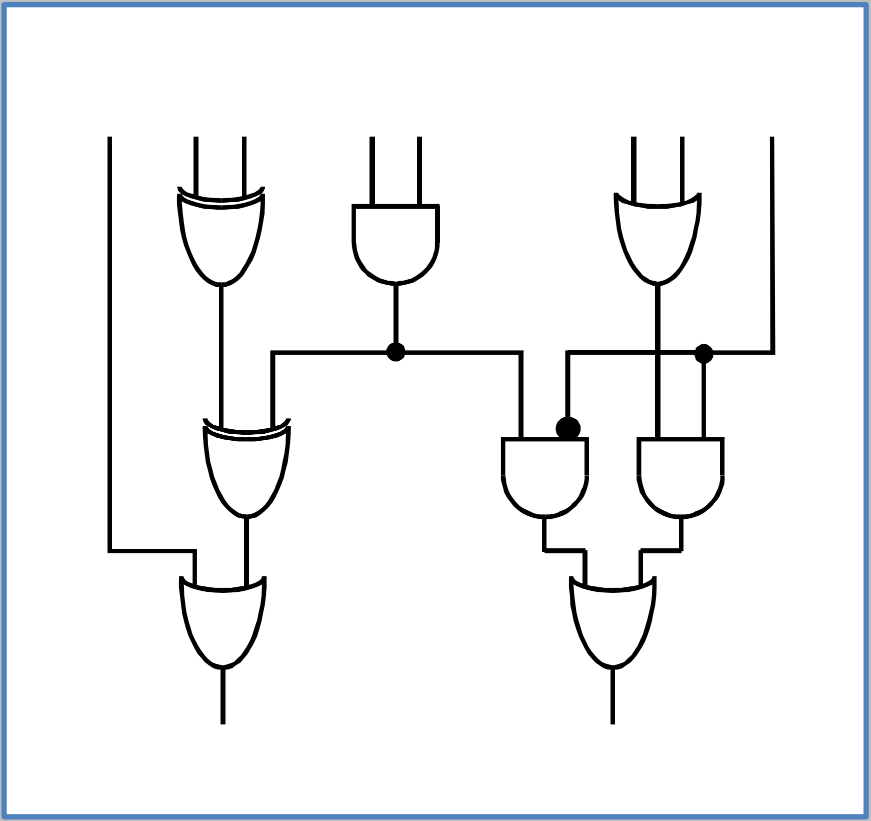
Implementation of Boolean functions

- In this way, implementations of all required **basic operations** are designed.

  These comprise the cells of a **cell library**.

- More complex functions:
  „Composition“ of these basic operations

Implementation of Boolean functions: Example of a Boolean function $f \in \mathbf{B}_{8,2}$



Questions:

1. How to model circuits mathematically?

Syntax

2. Which Boolean function is computed by a given circuit?

- Concrete simulation
- Symbolic simulation

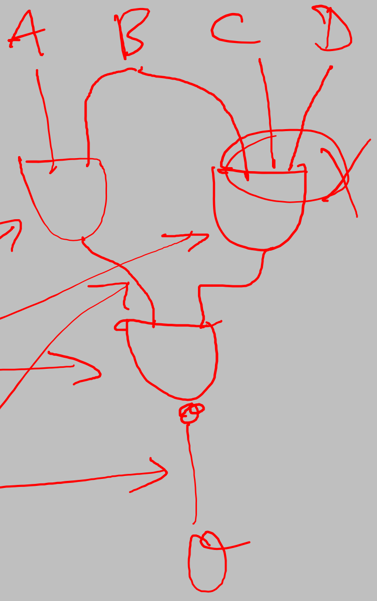
Semantics

Modeling circuits

Intuitively:

A **circuit** is a **directed graph** with some additional properties.

- DIRECTED GRAPH $G = (V, E)$
  - V = SET OF VERTICES
  - $E \subseteq V \times V$ OF EDGES

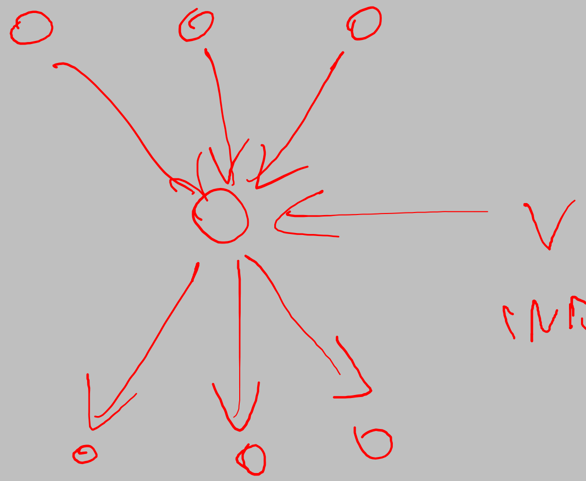


Modeling circuits (1/3)

- A **cell library** BIB $\subseteq$ $\mathbf{B}_n$ contains basic operations corresponding to basic gates
- A 5-tuple $C = (X_n, G, type, IN, Y_m)$ is called **circuit** with n inputs and m outputs (for library BIB) *iff*
  - $X_n = (x_1, \ldots, x_n)$ is a finite sequence of inputs.
  - $G = (V, E)$ is a directed acyclic graph (DAG) with $\{0, 1\} \cup \{x_1, \ldots, x_n\} \subseteq V$ and $E \subseteq V \times V$.
  - The set $I = V \setminus (\{0, 1\} \cup (x_1, \ldots, x_n))$ is called the **set of gates**.

Modeling circuits (2/3)

- The mapping $type : I \rightarrow BIB$ assigns a cell type $type(v) \in BIB$ to each gate $v \in I$.
- For each gate $v \in I$ with $type(v) \in \mathbf{B}_k$ we have $indeg(v) = k$.
  For $v \in V \setminus I = \{0, 1\} \cup \{x_1, ..., x_n\}$ we have $indeg(v) = 0$.



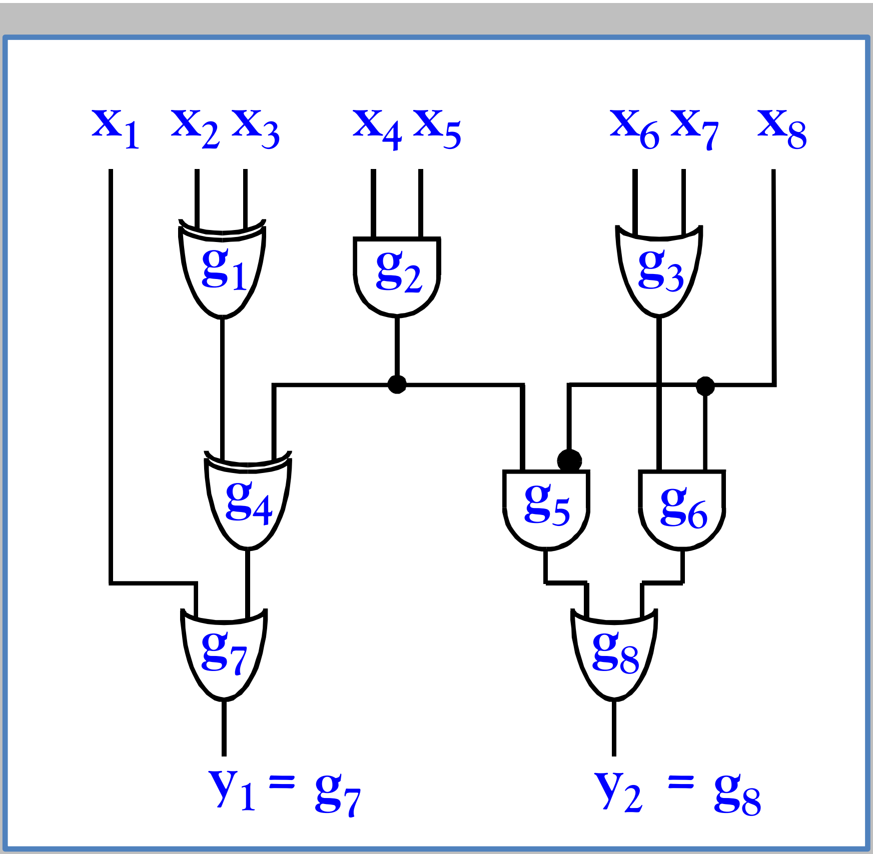
$INDEG(V) = |\{(V',V) \in E\}|$

$INDEG(V) = 3$

Modeling circuits (3/3)

- The mapping $IN : I \to V^*$ determines the order of the incoming edges, i.e., if $indeg(v) = k$ then $IN(v) = (v_1, \ldots, v_k)$ with $\forall 1 \leq i \leq k: (v_i, v) \in E$.
- The sequence $Y_n = (y_1, \ldots, y_n)$ designates the nodes $y_i \in V$ as the circuit's outputs.

Example circuit



Inputs:

$X = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$

Outputs:

$Y = (g_7, g_8)$

Gates:

$I = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8\}$

Edges of the graph:

$E = \{(x_1,g_7), (x_2,g_1), (x_3,g_1), (x_4,g_2)$
$(x_5,g_2), (x_6,g_3), (x_7,g_3), (x_8,g_5),$
$(x_8,g_6), (g_1,g_4), (g_2,g_4), \ldots\}$

Types of gates:

$type(g_1) = type(g_4) = XOR_2$

$type(g_2) = type(g_6) = AND_2 \ldots$

Order of the incoming edges:

$IN(g_1)=(x_2, x_3)$

$IN(g_4)=(g_1, g_2) \ldots$

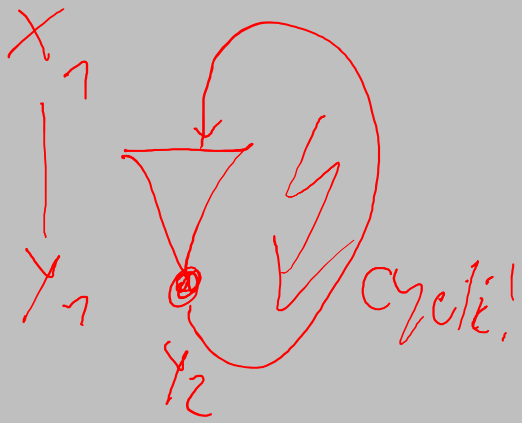
Semantics of circuits (1/2)

- Let $C = (X_n, G, typ, IN, Y_m)$ be a **circuit** for the **cell library BIB**.
- Let $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbf{B}^n$ be an **input valuation**.
- A valuation $\Phi_{C,\alpha} : V \rightarrow \{0, 1\}$ for all nodes $v \in V$ is given via the following definitions:
  - $\Phi_{C,\alpha}(x_i) = \alpha_i \ \forall 1 \leq i \leq n$
  - $\Phi_{C,\alpha}(0) = 0, \Phi_{C,\alpha}(1) = 1$
  - If $v \in I$ with

    $type(v) = g \in \mathbf{B}_k$ and $IN(v) = (v_1, \ldots, v_k)$,

    then

    $\Phi_{C,\alpha}(v) := g(\Phi_{C,\alpha}(v_1), \ldots, \Phi_{C,\alpha}(v_k))$.



Why is $\Phi_{C,\alpha}(v)$ well-defined?

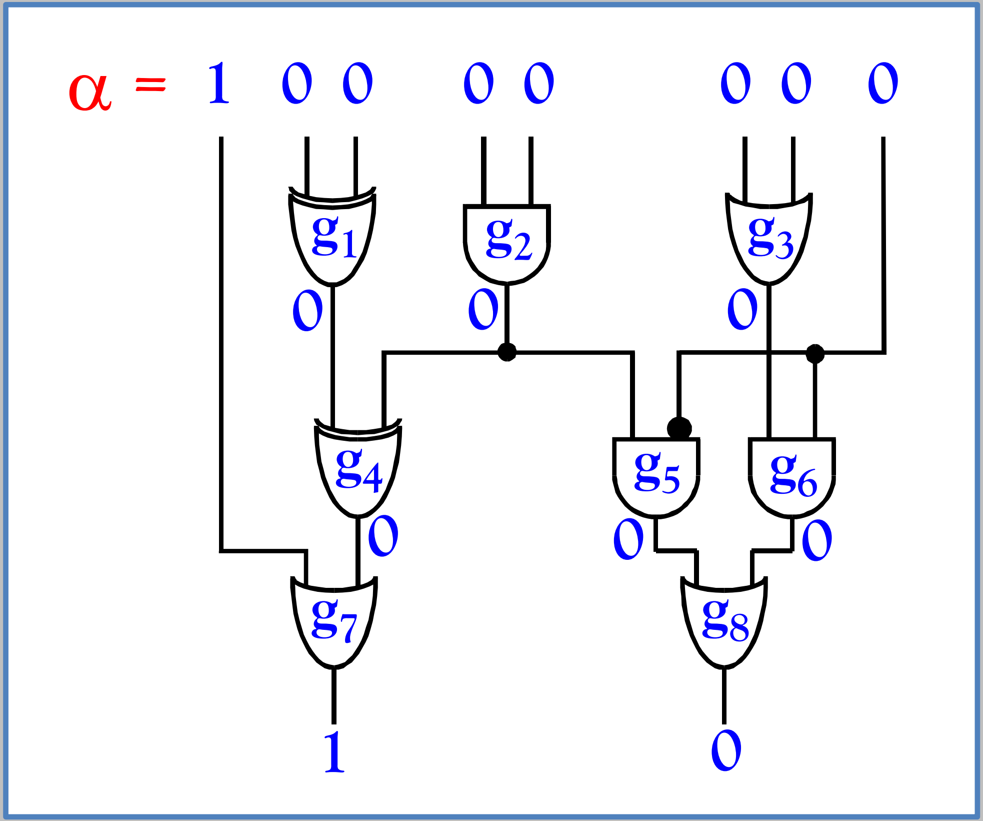
Because the underlying graph G is **acyclic**!

$\Phi_{C,\alpha}(y_2) = NOT(\Phi_{C,\alpha}(y_2))$

Semantics of circuits (2/2)

- Then $(\Phi_{C,\alpha}(y_1), \ldots, \Phi_{C,\alpha}(y_m))$ is the output valuation of the circuit under the input valuation $\alpha = (\alpha_1, \ldots, \alpha_n)$.
- The computation of $\Phi_{C,\alpha}$ under the input valuation α is called **simulation** of C under valuation α.

Example: Simulation



ctype(g_5) =

~~= AND(~~

$\lambda x, y.\ \mathrm{AND}(x, \mathrm{NOT}(y))$

Which Boolean function does a circuit compute?

Definition:

The function computed at a node *v*

$$\psi(v) : \mathbf{B}^n \to \mathbf{B}$$

is defined as

$$\psi(v)(\alpha) := \Phi_{C,\alpha}(v)$$

for an arbitrary $\alpha \in \mathbf{B}_n$.

Definition:

The function computed by circuit C is

$$f_C := (\psi(y_1), \ldots, \psi(y_m))$$

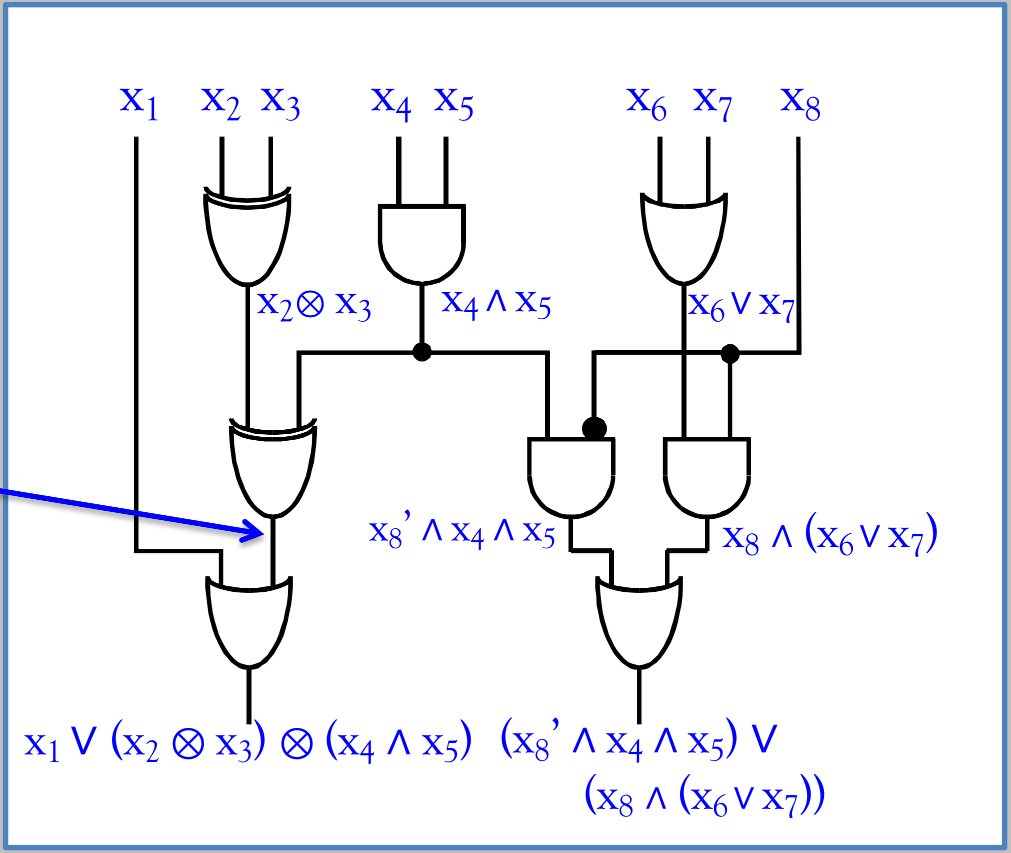
Symbolic simulation

- **Symbolic simulation** does not simulate a circuit for fixed Boolean inputs. Rather it simulates the circuit on Boolean variables.

- In this way it determines the **Boolean expression** representing the **Boolean function** computed by a circuit

Example: Symbolic simulation

$(x_2 \otimes x_3) \otimes (x_4 \wedge x_5)$


x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
$x_2 \otimes x_3$
$x_4 \wedge x_5$
$x_6 \vee x_7$
$x_8' \wedge x_4 \wedge x_5$
$x_8 \wedge (x_6 \vee x_7)$
$x_1 \vee (x_2 \otimes x_3) \otimes (x_4 \wedge x_5)$
$(x_8' \wedge x_4 \wedge x_5) \vee (x_8 \wedge (x_6 \vee x_7))$

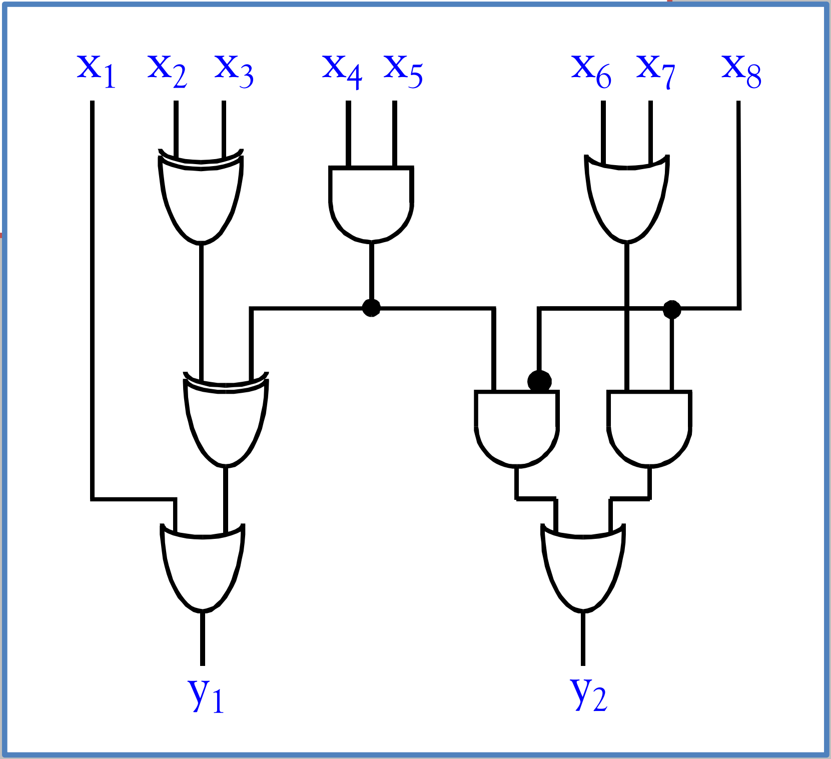
Brainstorming: Cost and Speed

What are reasonable measures of

(a) Cost and

(b) Speed

of circuits?



Cost of circuits

Definition (Cost):

The **hardware cost** C(C) of a circuit C is its number of gates $|I| = |V \setminus (\{0, 1\} \cup (x_1, \ldots, x_n))|$.

Remark:

- Circuits are defined based on a cell library BIB
  → Cost depends on the choice of the library.
- If not stated otherwise, in the following we will use the **standard library** STD:

  STD := {NOT, AND, OR, EXOR, NAND, NOR}

Speed of a circuit

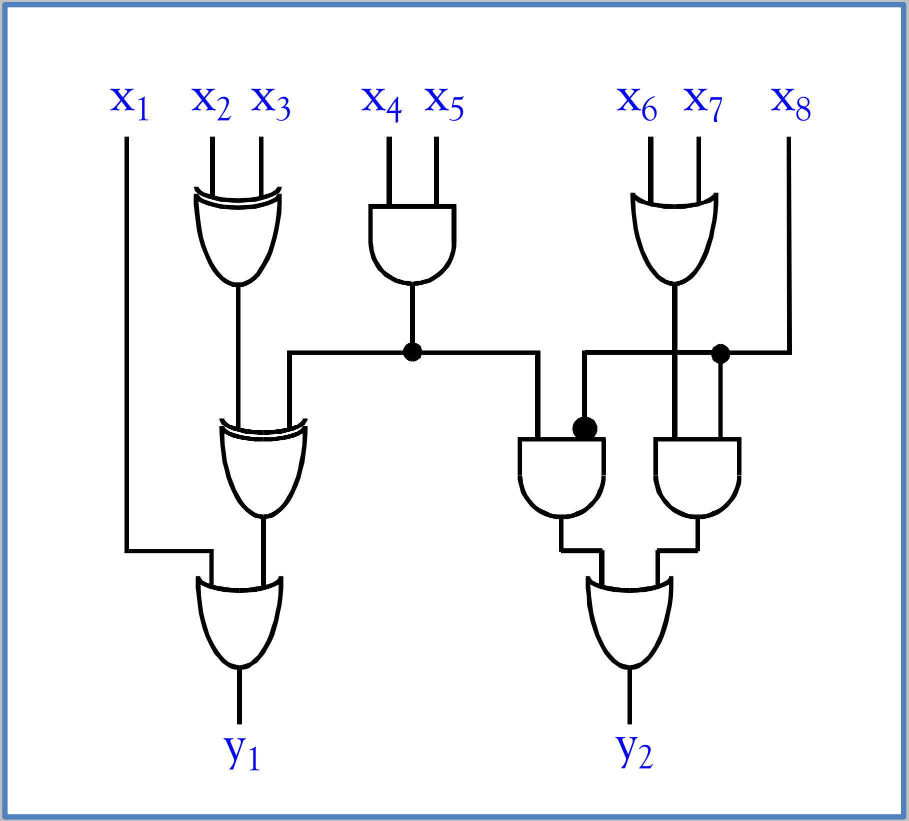
Definition (Depth):

The **depth** depth(C) of a circuit C is the **maximal number of gates on a path** from an arbitrary input x_i to an arbitrary output y_j of C.

Remark:

- Depth is only a reasonable indicators of a circuit's speed if the switching speed of each gate in the library is approximately the same.

Example: Cost and depth of circuits



Cost: 8

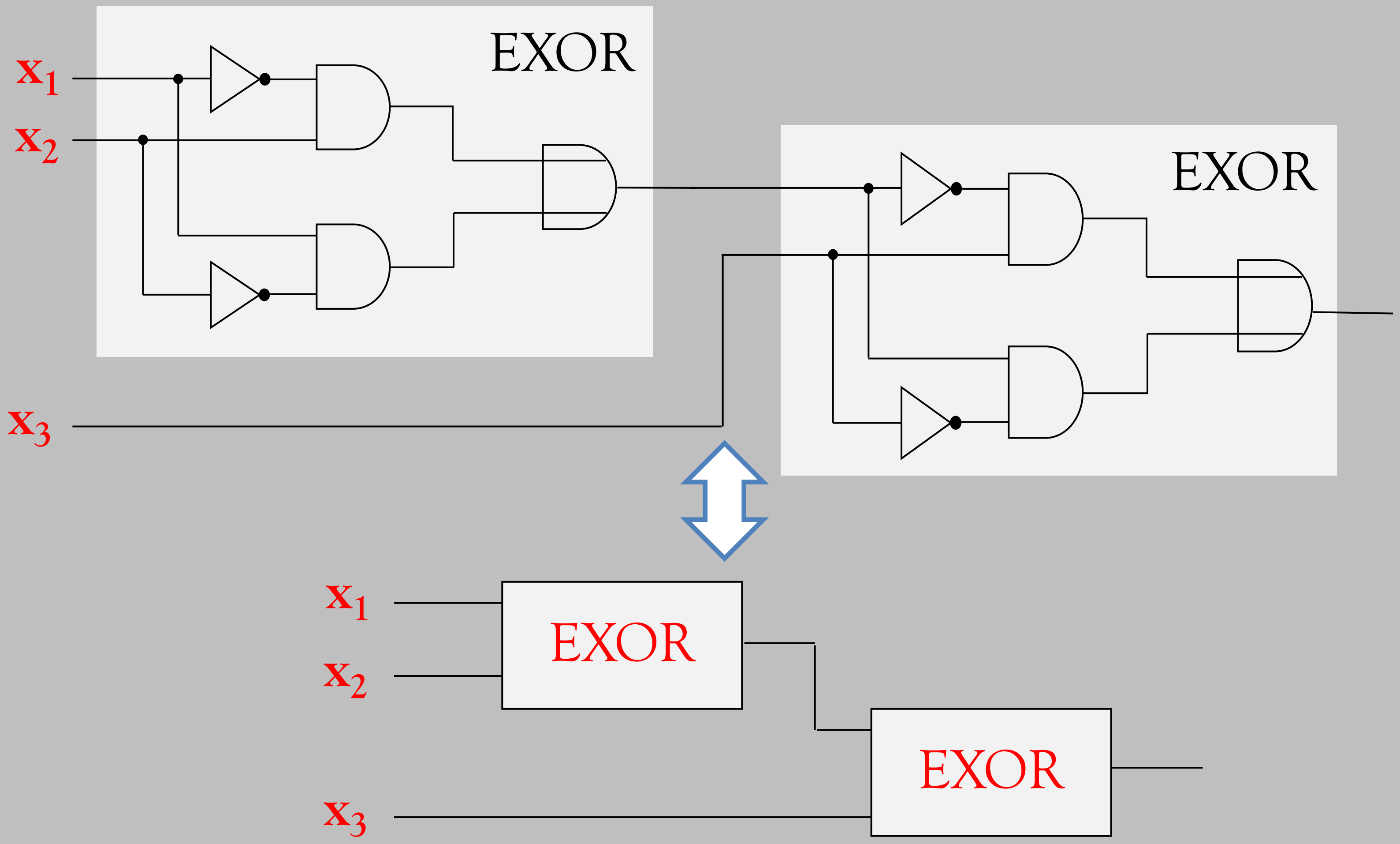
Depth: 3

Hierarchical circuits

In **hierarchical circuits**, subcircuits are represented by symbols.

The corresponding ("flat") circuit is obtained by replacing the symbols by their defining subcircuits.

Example Hierarchical circuits



Circuits vs Boolean functions

Every circuit computes a Boolean function.

But can **every** Boolean function be computed by a circuit?

Circuits vs Boolean functions

Theorem:

Let $f \in B_{n,m}$.

Then there is a circuit that computes f.

Reminder:

Lemma:

For every Boolean function $f \in B_{n,1}$ there is a Boolean expression that describes f.

Circuits vs Boolean expressions

Lemma:

For every Boolean expression $e \in BE(X_n)$ there is a circuit $C = (X_n, G, typ, IN, Y_m)$, *such that* $\psi(e) = f_C$.

Proof:
By induction over the structure of the Boolean expression.

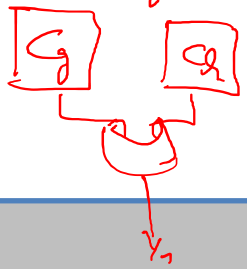
Recapitulation: Boolean expressions

Definition:

The set $BE(X_n)$ of fully parenthesized Boolean expressions over X_n is the smallest subset of A^*, inductively defined as follows:

- The elements 0 and 1 are Boolean expressions 0 1
- The variables $\mathbf{x_1}, \ldots, \mathbf{x_n}$ are Boolean expressions x_n
- Let **g** and **h** be Boolean expressions. Then so is their Disjunction **(g + h)**, their Conjunction **(g · h)**, and their Negation **(~g)**.

BY I.H., THERE ARE CIRCUITS C_g AND C_h, s.t. $\psi(g) = f_{C_g}$
$\psi(h) = f_{C_h}$



Circuits vs Boolean functions

Theorem:

Let $f \in B_{n,m}$.

Then there is a circuit that computes f.

Proof:

Case 1: $f \in \mathbf{B}_n = \mathbf{B}_{n,1}$. $\exists e \in BE(X_n)$, that computes f.

The theorem then directly follows from the previous lemma.

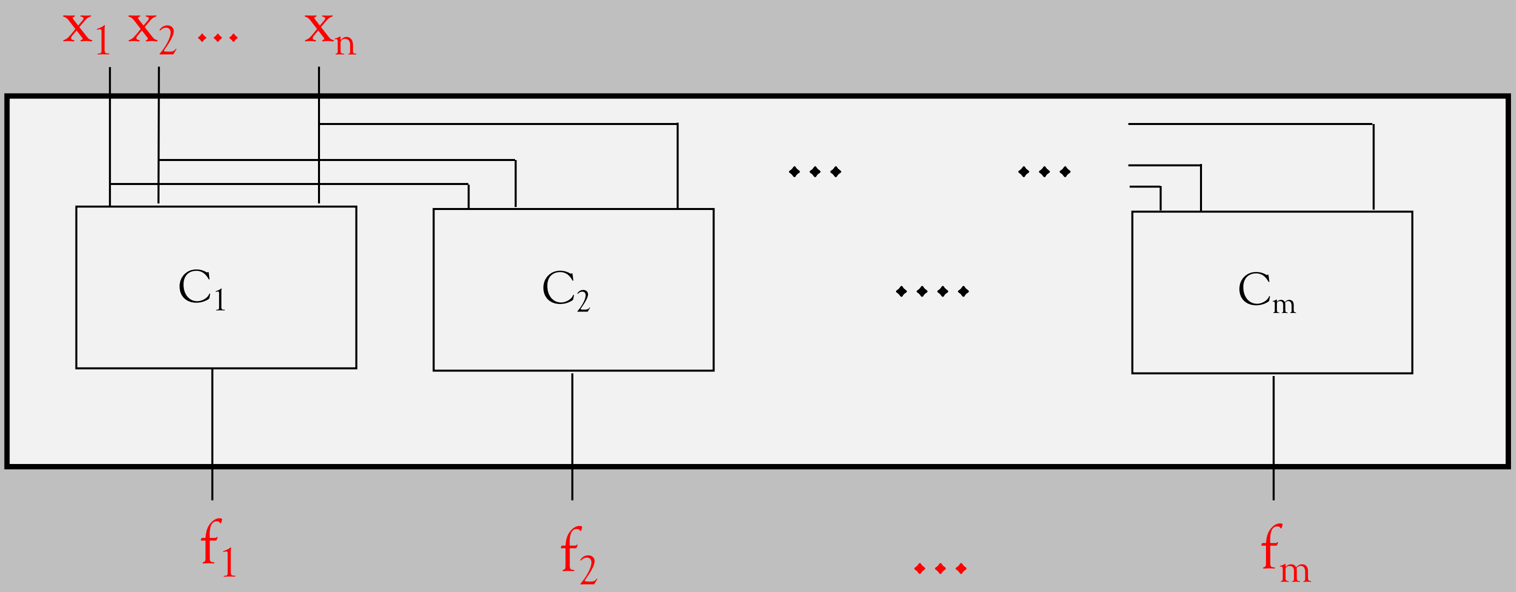
Case 2: $f \in \mathbf{B}_{n,m}$, $m \geq 2$.

Interpret $f : \mathbf{B}_{n,m} = \mathbf{B}^n \rightarrow \mathbf{B}^m$ as a sequence of functions $(f_1, \ldots, f_m)$ with $f_i : \mathbf{B}_n \rightarrow \mathbf{B}$.

Construct a circuit for each f_i.

Compose the circuits (see the following illustration).

Construction of a circuit for a Boolean function from $B_{n,m.}$



Example: Generalized EXOR

Given:

Function $\text{exor}_{16} \in \mathbf{B}_{16}$ with

$$exor_{16}(x_1, \dots, x_{16}) = \left(\sum_{i=1}^{16} x_i\right) \bmod 2 = 1 \text{ if number of } x_i \text{ with } x_i = 1 \text{ is odd}$$

Wanted:

Circuit implementation for exor_{16}.

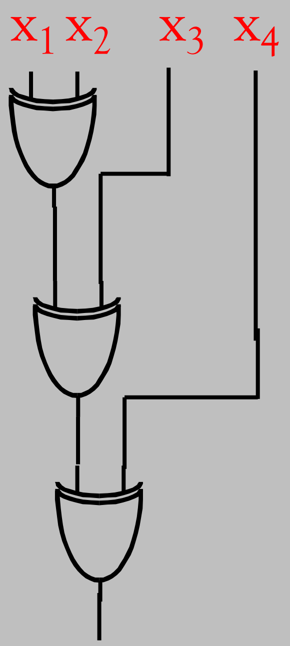
Assumption: exor_2 is an element of our cell library.

Observations:

1. exor_{16} can be constructed from several $\otimes = \text{exor}_2$.
2. $\otimes$ is an associative operation!

Generalized EXOR

Implementation of $exor_4$:



$(((x_1 \otimes x_2) \otimes x_3) \otimes x_4)$

Depth: 3
Cost: 3

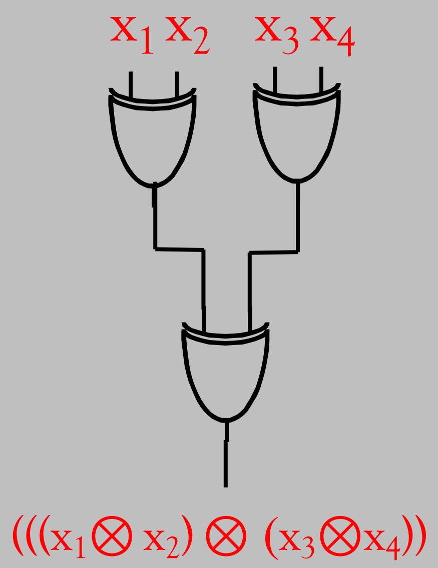
Can we do better?

Idea: Make use of associativity:

$(((x_1 \otimes x_2) \otimes x_3) \otimes x_4) = (((x_1 \otimes x_2) \otimes (x_3 \otimes x_4))$

Generalized EXOR

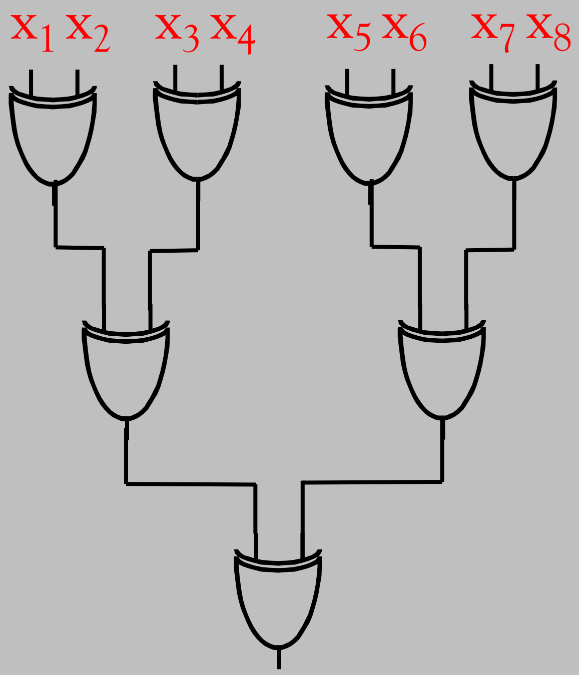
Better implementation of $exor_4$:



Depth: 2
Cost: 3

Generalized EXOR

Better implementation of $exor_8$:

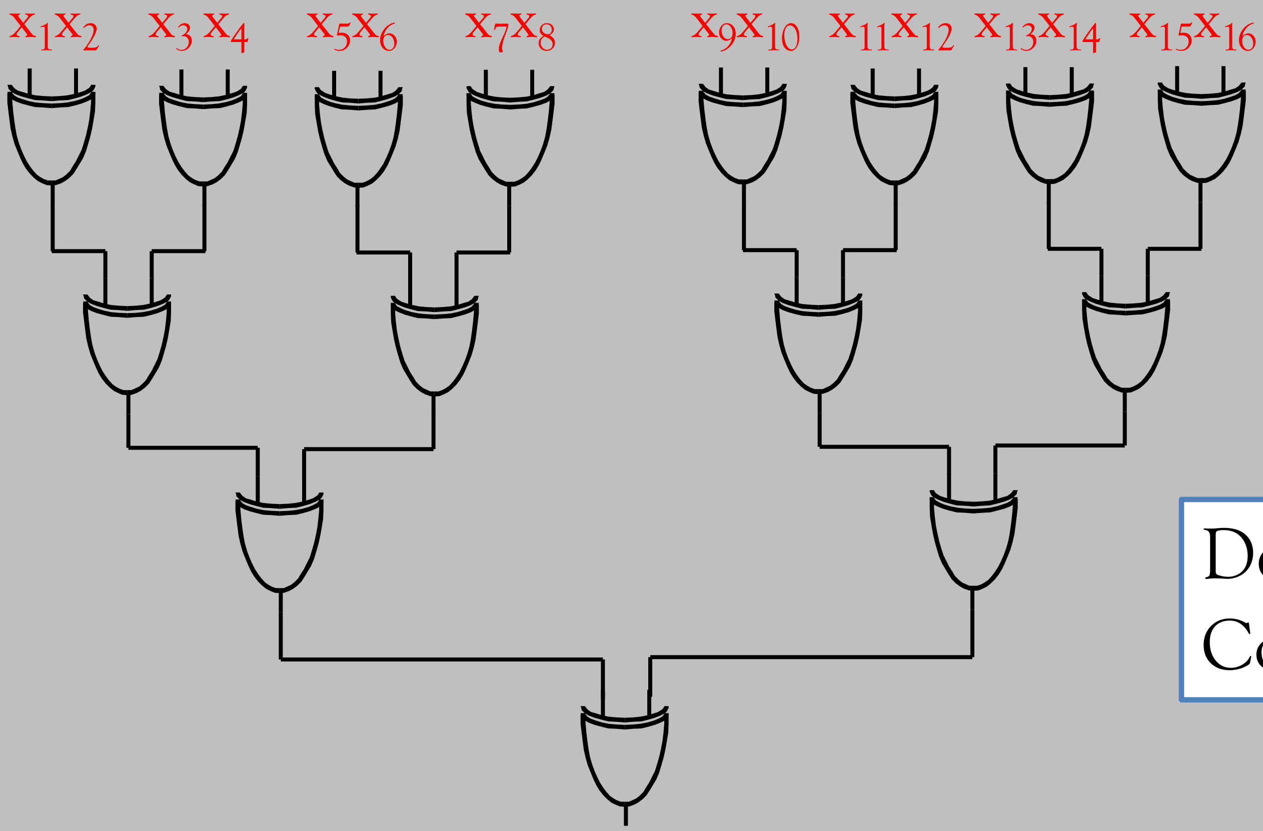


$(((x_1 \otimes x_2) \otimes (x_3 \otimes x_4)) \otimes (((x_5 \otimes x_6) \otimes (x_7 \otimes x_8))$

Depth: 3
Cost: 7

Generalized EXOR

Better implementation of exor_{16}:

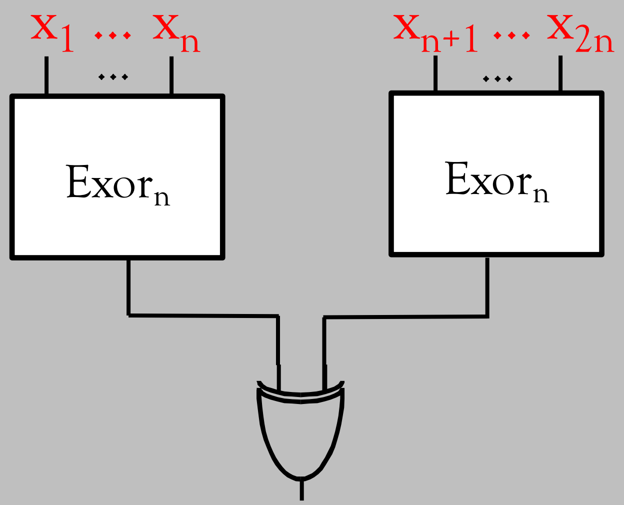


Depth: 4
Cost: 15

How do cost and depth depend on n for exor_n?

Recursive construktion of generalized EXOR

Implementation of $exor_{2n}$:



$depth(exor_{2n}) = depth(exor_n)+1$
$depth(exor_1) = 0$

→ $depth(exor_n) = \log_2 n$

$C(exor_{2n}) = 2 \cdot C(exor_n)+1$
$C(exor_1) = 0$

→ $C(exor_n) = n-1$

Efficient implementation of **arbitrary** associative operations

Lemma:

The function $x_1 \circ x_2 \dots \circ x_n$ can be implemented using $\circ$ gates with 2 inputs in a circuit of depth $\lceil \log_2 n \rceil$.

Proof by induction over n.

Two-level normal form of EXOR_{16}

Question: How large is the smallest Boolean polynomial of exor_{16}?

Answer: 2^{15} monomials with 16 literals each!

Question: How large it the smallest Boolean polynomial for exor_n?

Answer : $2^{n\text{-}1}$ monomials with n literals each!

Exponentially higher cost than the multi-level implementation!

Cost of the implementation of Boolean expressions via circuits

Define the cost $C(E)$ of a Boolean expression E to be the number of operations in the expression.

Theorem:

For every Boolean expression $e \in BE(X_n)$ there is a circuit $C = (X_n, G, typ, IN, Y_m)$, *such that* $\psi(e) = f_C$ and $C(C) \leq C(E)$.

Follows from proof of earlier lemma.

Reusing subcircuits can sometimes help reduce the cost.

Cost of the implementation of Boolean functions via circuits

Theorem:

For every $f \in \mathbf{B}_n$ there is a circuit C implementing f, s.t. $C(C) \leq n2^{n+1}-1$ and $depth(C) \leq n+\lceil \log_2 n \rceil+1$.

Proof sketch:

(Cost:) A function $f \in \mathbf{B}_n$ has at most 2^n minterms.
Every minterm can be implemented using $2n-1$ gates.
The disjunction of all minterms can be implemented using at most 2^n-1 gates.

(Depth:) Every minterm can be implemented in depth $\lceil \log_2 n \rceil+1$.
The disjunction can be implemented in depth n ($= \log_2 2^n$).

Summary

Circuits *implement* arbitrary Boolean functions from $B_{n,m}$.

Optimal Boolean polynomials can be much larger than corresponding multi-level circuits: **exponential differences** are possible!

Outlook

There are **algorithms** to compute **optimal multi-level circuits**

- harder than computing minimal polynomials
- mostly heuristics, i.e., not guaranteed to be optimal
- not covered in this course
- *Here*: Circuits for special functions,
  in particular arithmetic