

# Digital Logic Systems

## Recitation 4: Propositional Logic and Logisim Software

Guy Even    Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

March 26, 2019

## Formulas vs Functions

**Boolean Formulas** are strings made of variables, constants and connectives. Whereas **Boolean Functions** are mathematical objects described by truth tables. We will learn how to express boolean functions by formulas.

## $\tau$ vs $\hat{\tau}$

$\tau(x): U \rightarrow \{0, 1\}$  is a simple truth assignment to a variable  $x$ .

$\hat{\tau}(\phi): \mathcal{BF} \rightarrow \{0, 1\}$  is a truth value of a formula  $\phi$ , which is evaluated by applying  $\tau$  to its variables and computing the EVAL algorithm.

## Equivalent connective symbols

$$(A + B) = (A \vee B) = (A \text{ OR } B)$$

$$(A \cdot B) = (A \wedge B) = (A \text{ AND } B)$$

$$(\neg B) = (\text{NOT}(B)) = (\bar{B})$$

$$(A \text{ XOR } B) = (A \oplus B)$$

## Parantheses

$$((A \vee C) \wedge (\neg B)) = ((A + C) \cdot (\bar{B}))$$

We sometimes omit parentheses from formulas if their parse tree is obvious. When parenthesis are omitted, one should use precedence rules as in arithmetic, e.g.,  $a \cdot b + c \cdot d = ((a \cdot b) + (c \cdot d))$ .

# Useful tautologies

$$\begin{array}{ll} X + 0 \leftrightarrow X & X \cdot 1 \leftrightarrow X \\ X + 1 \leftrightarrow 1 & X \cdot 0 \leftrightarrow 0 \end{array}$$

• כללי האפס והיחידה:

$$X + X \leftrightarrow X \qquad X \cdot X \leftrightarrow X$$

• כללי הכפילות:

$$X + \bar{X} \leftrightarrow 1 \qquad X \cdot \bar{X} \leftrightarrow 0$$

• כללי ההיפוך:

$$x + xy \leftrightarrow x$$

• כלל הבליעה הראשון:

$$x + \bar{x}y \leftrightarrow x + y$$

• כלל הבליעה השני:

$$1. \quad \overline{\bar{x} \cdot \bar{y}} \leftrightarrow \bar{x} + \bar{y}$$

• חוקי דה-מורגן (לשני משתנים):

$$2. \quad \overline{x + y} \leftrightarrow \bar{x} \cdot \bar{y}$$

# Tautologies

## Example

Prove that the following formulas are tautologies: (i) addition:  $\phi_1 \triangleq (X \rightarrow (X + Y))$ , and (ii) simplification:  $\phi_2 \triangleq ((X \cdot Y) \rightarrow X)$ .

## Proof.

The proof is by truth tables, The following figure depicts the tables of both formulas. Note that the row that represents  $\hat{\tau}_v(\phi_i)$  is a constant Boolean function, i.e.,  $\forall v \in \{0, 1\}^2 : \hat{\tau}_v(\phi_i) = 1$ .

| $X$ | $Y$ | $X + Y$ | $\phi_1$ | $X$ | $Y$ | $X \cdot Y$ | $\phi_2$ |
|-----|-----|---------|----------|-----|-----|-------------|----------|
| 0   | 0   | 0       | 1        | 0   | 0   | 0           | 1        |
| 1   | 0   | 1       | 1        | 1   | 0   | 0           | 1        |
| 0   | 1   | 1       | 1        | 0   | 1   | 0           | 1        |
| 1   | 1   | 1       | 1        | 1   | 1   | 1           | 1        |

**Table:** The truth tables of the addition and the simplification tautologies.



# The Logisim Software

- Can be downloaded from:  
<http://www.cburch.com/logisim/>
- **Input ports** to assign inputs to your circuit.
- **Output ports** to gain the results.
  - Light green = '1'.
  - Dark green = '0'.
- **Combinational Gates** (AND,MUX,XOR,...) implement boolean functions.
- **Project→Analyze Circuit** - generate the truth tables
- **Modular Design** - We compose Boolean functions to “construct” new ones. Use multiple circuits hierarchically. You will learn this principle as **Substitution**.
- **Labels** must be assigned to I/O ports!
- “Minimization heuristics” ...can be found in the book.

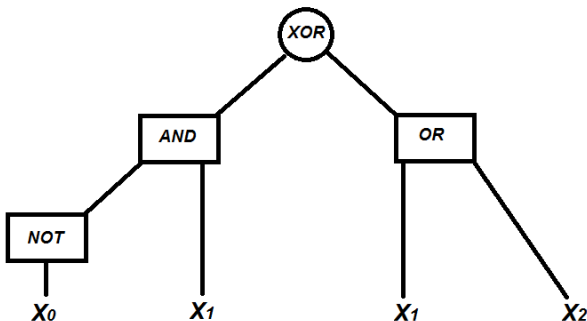
## Example

Consider the following boolean formula  $\phi$ :

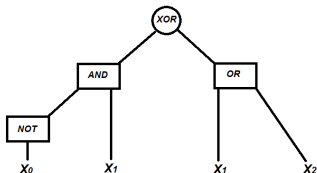
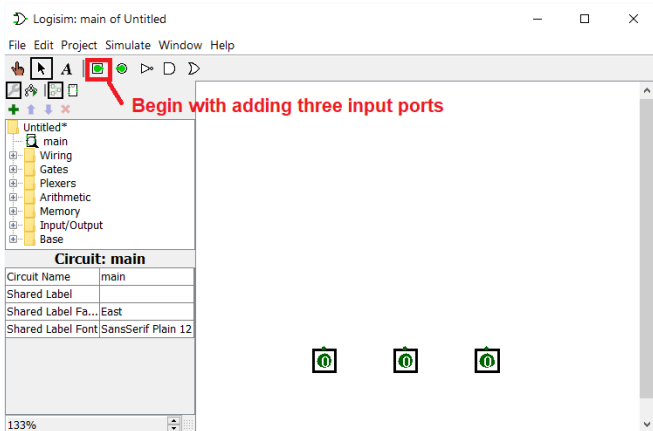
$$\phi = (x_2 \vee x_1) \oplus (\neg x_0 \wedge x_1)$$

Implement the boolean function  $B_\phi : \{0,1\}^3 \rightarrow \{0,1\}$  that corresponds to the formula  $\phi$ .

## Parse tree of a formula $\phi$



# Logisim Example - $B_\phi$ implementation





# Logisim Example - $B_\phi$ implementation

Logisim: main of Untitled

File Edit Project Simulate Window Help

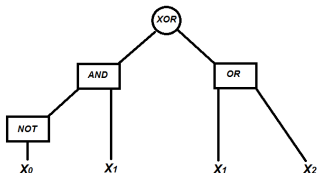
Selection: Pin

|                |                    |
|----------------|--------------------|
| Facing         | North              |
| Output?        | No                 |
| Data Bits      | 1                  |
| Three-state?   | No                 |
| Pull Behavior  | Unchanged          |
| Label          | X0                 |
| Label Location | West               |
| Label Font     | SansSerif Plain 12 |

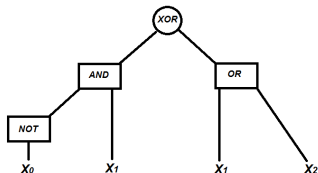
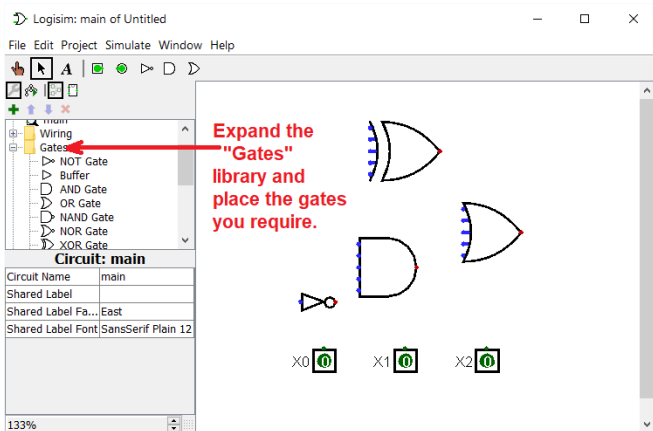
133%

Click on a port

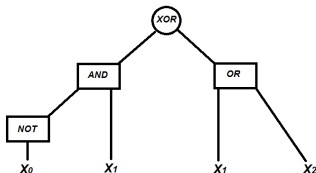
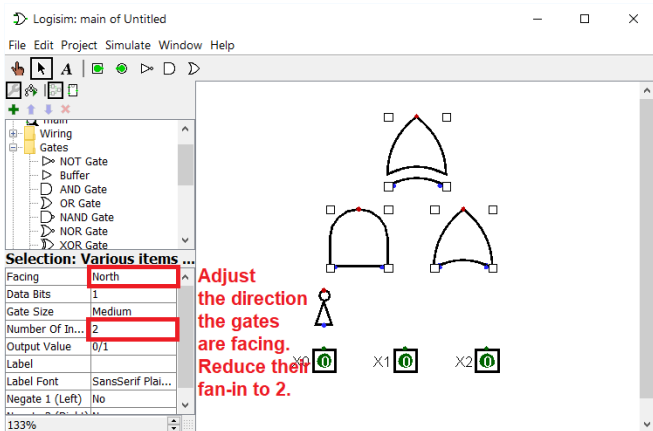
then modify its attributes



# Logisim Example - $B_\phi$ implementation



# Logisim Example - $B_\phi$ implementation



# Logisim Example - $B_\phi$ implementation

Logisim: main of Untitled

File Edit Project Simulate Window Help

Selection: Pin

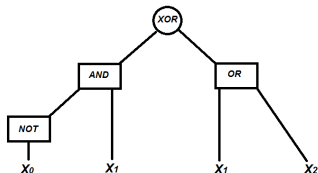
|                |                    |
|----------------|--------------------|
| Facing         | South              |
| Output?        | Yes                |
| Data Bits      | 1                  |
| Three-state?   | Yes                |
| Pull Behavior  | Unchanged          |
| Label          | Y                  |
| Label Location | East               |
| Label Font     | SansSerif Plain 12 |

133%

Add an output port

Give it a name

The screenshot shows the Logisim software interface. On the left is a component library with categories like 'Wiring' and 'Gates'. Below it is a 'Selection: Pin' configuration table. The table has fields for 'Facing', 'Output?', 'Data Bits', 'Three-state?', 'Pull Behavior', 'Label', 'Label Location', and 'Label Font'. The 'Label' field is highlighted with a red box and contains the text 'Y'. A red arrow points from the text 'Add an output port' to the output pin symbol in the circuit diagram. Another red arrow points from the text 'Give it a name' to the 'Label' field in the configuration table. The circuit diagram on the right shows a hierarchical structure of logic gates: a NOT gate connected to an AND gate, which is connected to an XOR gate. The XOR gate is also connected to an OR gate, which is connected to the output pin Y. The inputs to the gates are labeled X0, X1, and X2.





# Logisim Example - $B_\phi$ implementation

Logisim: main of Untitled

File Edit Project Simulate Window Help

Input/Output Base  
Poke Tool  
Edit Tool  
Select Tool  
Wiring Tool  
Text Tool  
Menu Tool  
Label

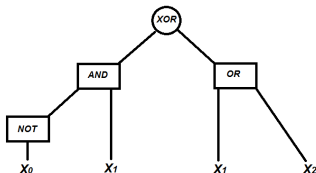
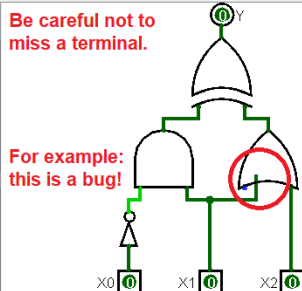
**Circuit: main**

|                    |                    |
|--------------------|--------------------|
| Circuit Name       | main               |
| Shared Label       |                    |
| Shared Label Fa... | East               |
| Shared Label Font  | SansSerif Plain 12 |

133%

Be careful not to miss a terminal.

For example: this is a bug!



# Logisim Example - $B_\phi$ implementation

Logisim: main of Untitled

File Edit Project Simulate Window Help

Input/Output Base

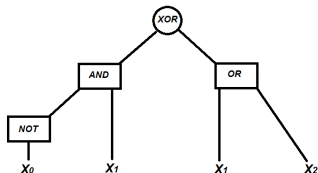
- Poke Tool
- Edit Tool
- Select Tool
- Wiring Tool
- Text Tool
- Menu Tool
- Label

**Pin**

|                |                    |
|----------------|--------------------|
| Facing         | North              |
| Output?        | No                 |
| Data Bits      | 1                  |
| Three-state?   | No                 |
| Pull Behavior  | Unchanged          |
| Label          | X1                 |
| Label Location | West               |
| Label Font     | SansSerif Plain 12 |

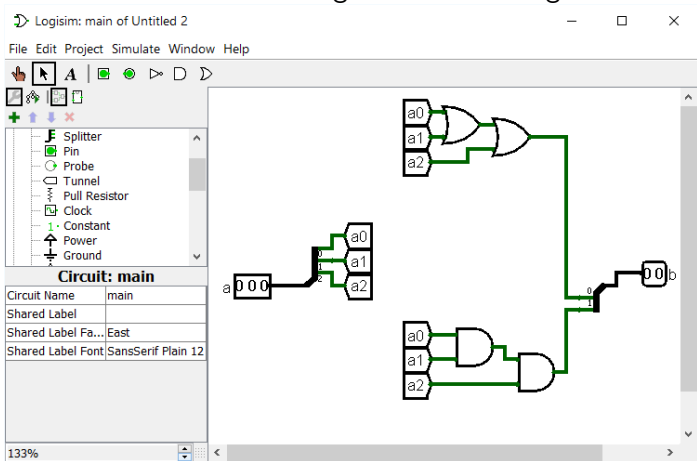
133%

(Simulation)  
Choose "poke tool" and click on input ports to toggle the logical value.



# Logisim - Wiring Library

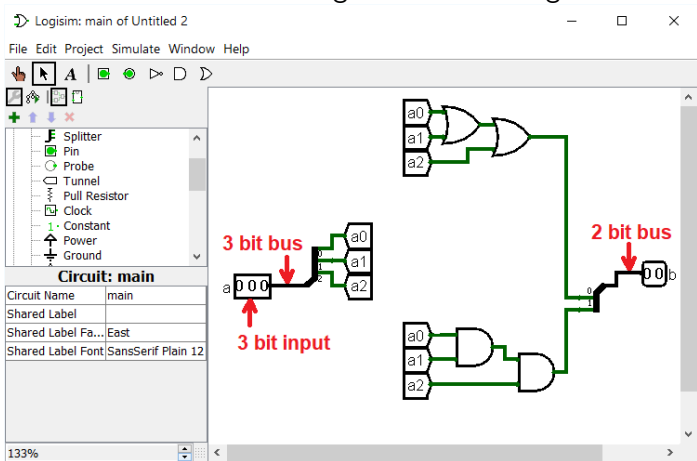
- 1 Bus - a set of parallel wires
- 2 Splitters - are used to split/collect a bus into individual wires
- 3 Tunnels - a.k.a. net labeling to avoid drawing connections.





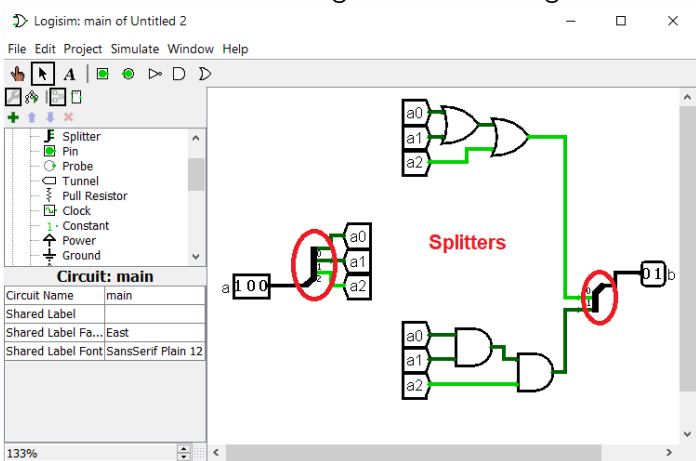
# Logisim - Wiring Library

- 1 Bus - a set of parallel wires
- 2 Splitters - are used to split/collect a bus into individual wires
- 3 Tunnels - a.k.a. net labeling to avoid drawing connections.



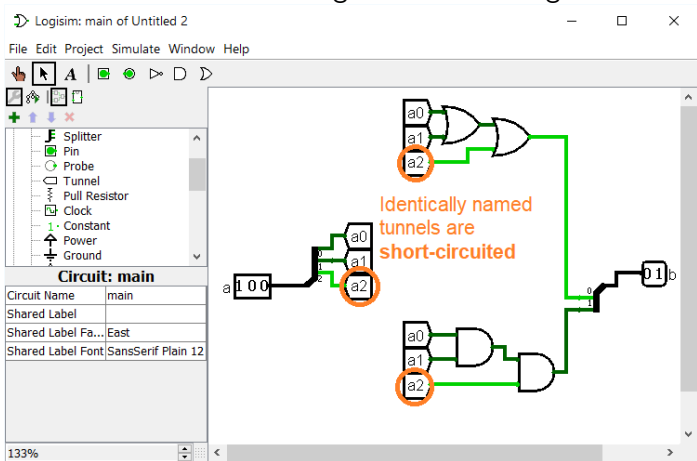
# Logisim - Wiring Library

- 1 Bus - a set of parallel wires
- 2 Splitters - are used to split/collect a bus into individual wires
- 3 Tunnels - a.k.a. net labeling to avoid drawing connections.

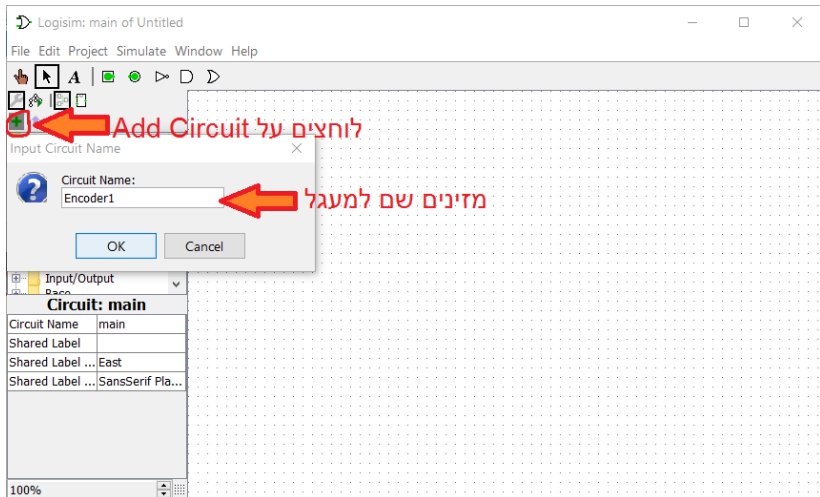


# Logisim - Wiring Library

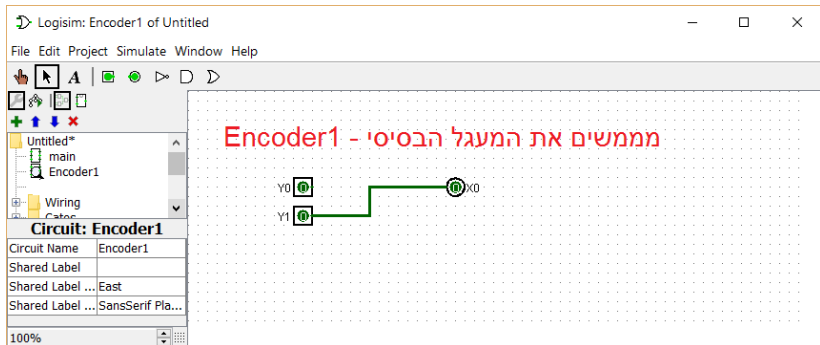
- ① Bus - a set of parallel wires
- ② Splitters - are used to split/collect a bus into individual wires
- ③ Tunnels - a.k.a. net labeling to avoid drawing connections.



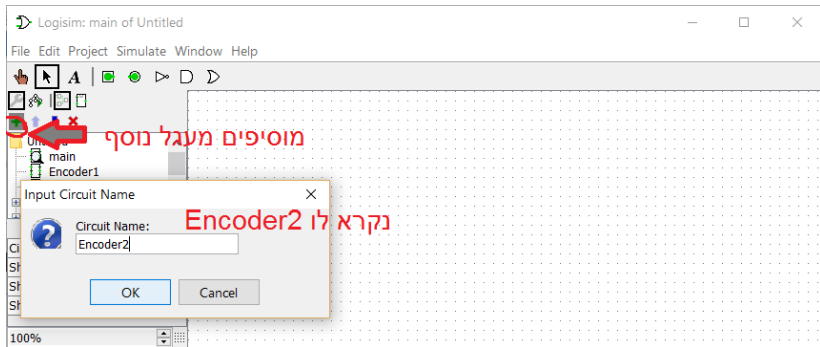
# Modular Design in Logisim



# Modular Design in Logisim



# Modular Design in Logisim



# Modular Design in Logisim

Logisim: Encoder2 of Untitled

File Edit Project Simulate Window Help

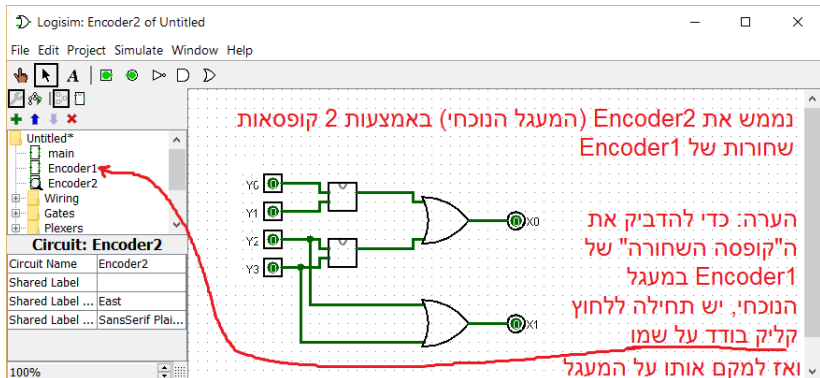
Y0 Y1 Y2 Y3 X0 X1

Encoder2

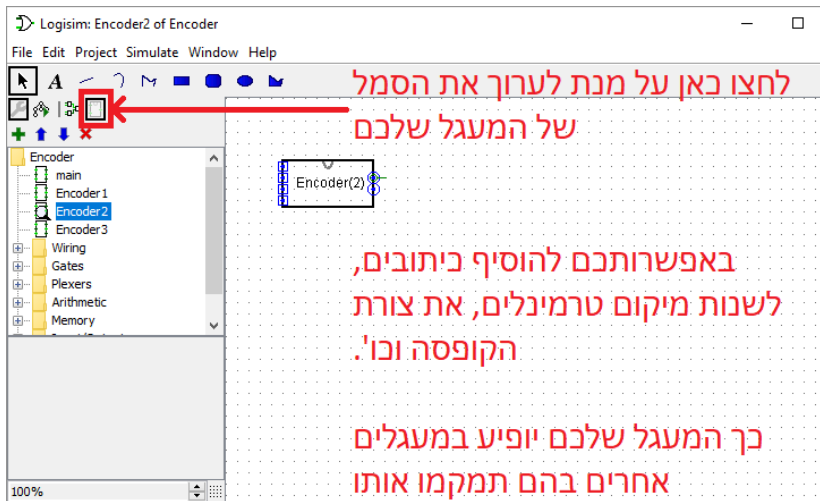
100%

נממש את Encoder2 (המעגל הנוכחי) באמצעות 2 קופסאות שחורות של Encoder1

הערה: כדי להדביק את ה"קופסה השחורה" של Encoder1 במעגל הנוכחי, יש תחילה ללחוץ קליק בודד על שמו ואז למקם אותו על המעגל



# Modular Design in Logisim



Logisim: Encoder2 of Encoder

File Edit Project Simulate Window Help

לחצו כאן על מנת לערוך את הסמל של המעגל שלכם

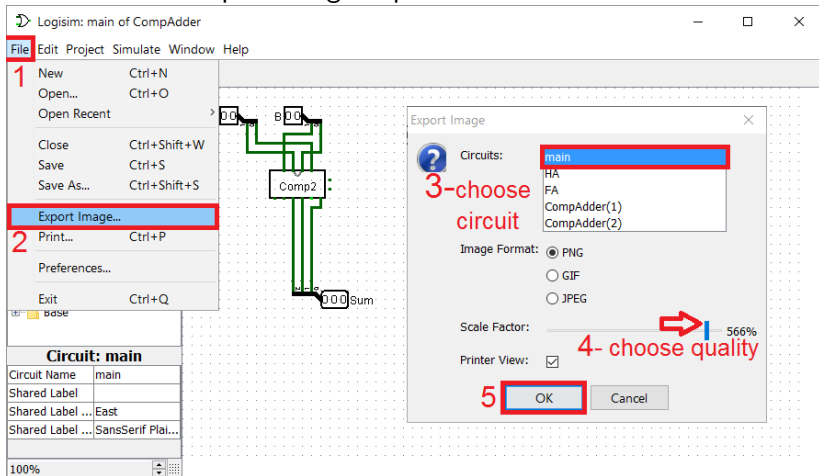
באפשרותכם להוסיף כיתובים, לשנות מיקום טרמינלים, את צורת הקופסה וכו'.

כך המעגל שלכם יופיע במעגלים אחרים בהם תמקמו אותו



# Exporting Circuit as an image without a truth table

Use the “File→Export Image” option



# Exporting Circuit as an image with a truth table

Open the “Project→Analyze Circuit” → “Table” then print screen:

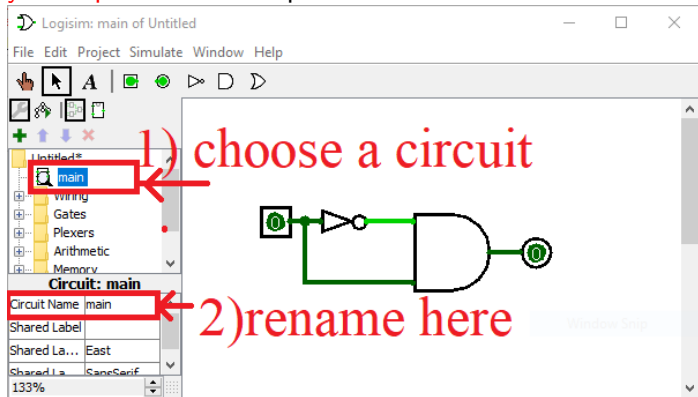
The screenshot displays the Logisim software interface. The main window shows a circuit diagram with four inputs (S0, X1, X0, S1) and one output (Z). The circuit consists of several logic gates: two AND gates, two OR gates, and one NOT gate. The inputs are connected to the gates as follows: S0 and X1 are connected to the first AND gate; X1 and X0 are connected to the second AND gate; S0 and X0 are connected to the first OR gate; S1 and X0 are connected to the second OR gate; and S1 is connected to the NOT gate. The outputs of the two AND gates and the output of the NOT gate are connected to the first OR gate, which produces the final output Z.

On the right side, a window titled "Combination..." is open, showing the "Table" tab. This window displays the truth table for the circuit, with columns for inputs S0, X1, X0, S1 and output Z. The table contains 16 rows of data, representing all possible combinations of the four inputs.

| S0 | X1 | X0 | S1 | Z |
|----|----|----|----|---|
| 0  | 0  | 0  | 0  | 0 |
| 0  | 0  | 0  | 1  | 0 |
| 0  | 0  | 1  | 0  | 1 |
| 0  | 0  | 1  | 1  | 0 |
| 0  | 1  | 0  | 0  | 0 |
| 0  | 1  | 0  | 1  | 0 |
| 0  | 1  | 1  | 0  | 0 |
| 0  | 1  | 1  | 1  | 0 |
| 1  | 0  | 0  | 0  | 0 |
| 1  | 0  | 0  | 1  | 0 |
| 1  | 0  | 1  | 0  | 0 |
| 1  | 0  | 1  | 1  | 0 |
| 1  | 1  | 0  | 0  | 0 |
| 1  | 1  | 0  | 1  | 1 |
| 1  | 1  | 1  | 0  | 1 |
| 1  | 1  | 1  | 1  | 0 |

# Adjusting the circuit name

In the project submission guidelines you are required to **rename your top module** to a specific name.



# Multi-bit IO ports

From now on, do not use multiple single-bit ports when indexing is required.

Inputs:  $a, b, c \in \{0, 1\}$ , then it is OK to place 3 single-bit inputs:

a 

b 

c 



Input:  $a[2:0] \in \{0, 1\}^3$ , then use a bus!

~~a0 ~~  
a1   
a2 

a 



# Complete Set of Connectives

- A Boolean formula expresses some Boolean function.
- We deal with the following question: Which sets of connectives enable us to express **every** Boolean function?

Recall the following definitions.

## Definition

A Boolean function  $B : \{0, 1\}^n \rightarrow \{0, 1\}$  is **expressible** by  $\mathcal{BF}(\{X_1, \dots, X_n\}, \mathcal{C})$  if there exists a formula  $p \in \mathcal{BF}(\{X_1, \dots, X_n\}, \mathcal{C})$  such that  $B = B_p$ .

## Definition

A set  $\mathcal{C}$  of connectives is **complete** if every Boolean function  $B : \{0, 1\}^n \rightarrow \{0, 1\}$  is expressible by  $\mathcal{BF}(\{X_1, \dots, X_n\}, \mathcal{C})$ .

## Theorem

*The set  $\mathcal{C} = \{\neg, \text{AND}, \text{OR}\}$  is a complete set of connectives.*

# $\mathcal{C} = \{\text{AND}, \text{OR}\}$ is not Complete Set of Connectives

## Theorem

*The set  $\mathcal{C} = \{\text{AND}, \text{OR}\}$  is not a complete set of connectives.*

## Proof.

- We prove that the Boolean function NOT is not expressible by  $\mathcal{BF}(\{X_1\}, \mathcal{C})$ .
- How? we prove that every  $\varphi \in \mathcal{BF}(\{X_1\}, \mathcal{C})$ ,  $B_\varphi$  is either the function 0, 1, or  $I$ , where  $I$  is the identity function.
- Proof by a complete induction on the size of the parse tree of the Boolean formula  $\varphi$  (next slide).
- Since NOT is not the function 0, 1, or  $I$ , it follows that NOT is not expressible by  $\mathcal{BF}(\{X_1\}, \mathcal{C})$ , as required.



## $\mathcal{C} = \{\text{AND}, \text{OR}\}$ is not Complete Set of Connectives

Proof by a complete induction on the size of the parse tree of a formula  $\varphi$ .

- **Base:** For trees of a size 1,  $\varphi$  can be one of the following options:  $\{0, 1, x\}$ . The corresponding  $B_\varphi(x)$  can be either const 0 or const 1 or the identity function  $x$ .
- **Induction Hypothesis:** For a formula  $\varphi$  with a parse tree of a size  $n$  and lower, the corresponding  $B_\varphi(x)$  can be either const 0 or const 1 or the identity function  $x$ .

# $\mathcal{C} = \{\text{AND}, \text{OR}\}$ is not Complete Set of Connectives

Proof by a complete induction on the size of the parse tree of a formula  $\varphi$ .

- **Induction Step:** We shall prove that for a formula  $\varphi$  (with a larger parse tree) the corresponding  $B_\varphi(x)$  can be either const 0 or const 1 or the identity function  $x$ .
- **Important tip:** We observe the construction of tree of  $\varphi$  and understand that we have to break the proof into 2 cases. Where each case corresponds to using a different connective.

- **Induction Step Proof:**

- ①  $\varphi = \varphi_1 \cdot \varphi_2$ . In this case,  $B_\varphi = B_{AND}(B_{\varphi_1}(x), B_{\varphi_2}(x))$

- ②  $\varphi = \varphi_1 + \varphi_2$ . In this case,  $B_\varphi = B_{OR}(B_{\varphi_1}(x), B_{\varphi_2}(x))$

The formulas  $\varphi_1, \varphi_2$  are smaller formulas and thus we can exploit the induction hypothesis. By induction hypothesis,  $B_{\varphi_1}(x), B_{\varphi_2}(x)$  can be either 0, 1 or  $x$ .

Let's take a look at all the possible functions  $B_\varphi \dots$



## $\mathcal{C} = \{\text{AND}, \text{OR}\}$ is not Complete Set of Connectives

**Table:** Given  $B_{\varphi_1}, B_{\varphi_2}$ , the following table describes what will be the  $B_{\varphi}$  for the both cases

| $B_{\varphi_1}$ | $B_{\varphi_2}$ | $B_{\varphi} = B_{AND}(B_{\varphi_1}, B_{\varphi_2})$ | $B_{\varphi} = B_{OR}(B_{\varphi_1}, B_{\varphi_2})$ |
|-----------------|-----------------|---|--|
| 0               | 0               | 0   | 0  |
| 0               | 1               | 0   | 1  |
| 0               | x               | 0   | x  |
| 1               | 0               | 0   | 1  |
| 1               | 1               | 1   | 1  |
| 1               | x               | x   | 1  |
| x               | 0               | 0   | x  |
| x               | 1               | x   | 1  |
| x               | x               | x   | x  |

We showed that in both cases,  $B_{\varphi} \in \{0, 1, x\}$  Therefore, we proved the induction step.

# $\mathcal{C} = \{\downarrow\}$ is a Complete Set of Connectives

## Example

Prove that  $\{\downarrow\}$  is a complete set of connectives. Where the connective  $\downarrow$  corresponds to a boolean function  $NOR_2(b_1, b_2)$ .

## Proof.

- We need to show that some other *complete set of connectives* can be expressed using only  $\downarrow$ .
- Let's express the complete set  $\{\neg, \vee, \wedge\}$  using  $\{\downarrow\}$

# $\mathcal{C} = \{\downarrow\}$ is a Complete Set of Connectives

## Example

Prove that  $\{\downarrow\}$  is a complete set of connectives. Where the connective  $\downarrow$  corresponds to a boolean function  $NOR_2(b_1, b_2)$ .

## Proof.

- We need to show that some other *complete set of connectives* can be expressed using only  $\downarrow$ .
- Let's express the complete set  $\{\neg, \vee, \wedge\}$  using  $\{\downarrow\}$
- $\neg x$  can be expressed by  $x \downarrow x$  or by  $x \downarrow 0$  (see truth table)

# $\mathcal{C} = \{\downarrow\}$ is a Complete Set of Connectives

## Example

Prove that  $\{\downarrow\}$  is a complete set of connectives. Where the connective  $\downarrow$  corresponds to a boolean function  $NOR_2(b_1, b_2)$ .

## Proof.

- We need to show that some other *complete set of connectives* can be expressed using only  $\downarrow$ .
- Let's express the complete set  $\{\neg, \vee, \wedge\}$  using  $\{\downarrow\}$
- $\neg x$  can be expressed by  $x \downarrow x$  or by  $x \downarrow 0$  (see truth table)
- $x \wedge y \Leftrightarrow \neg\neg(x \wedge y) \Leftrightarrow \neg(\neg x \vee \neg y) \Leftrightarrow (\neg x \downarrow \neg y)$   
 $\Leftrightarrow (x \downarrow x) \downarrow (y \downarrow y)$

# $\mathcal{C} = \{\downarrow\}$ is a Complete Set of Connectives

## Example

Prove that  $\{\downarrow\}$  is a complete set of connectives. Where the connective  $\downarrow$  corresponds to a boolean function  $NOR_2(b_1, b_2)$ .

## Proof.

- We need to show that some other *complete set of connectives* can be expressed using only  $\downarrow$ .
- Let's express the complete set  $\{\neg, \vee, \wedge\}$  using  $\{\downarrow\}$
- $\neg x$  can be expressed by  $x \downarrow x$  or by  $x \downarrow 0$  (see truth table)
- $x \wedge y \Leftrightarrow \neg\neg(x \wedge y) \Leftrightarrow \neg(\neg x \vee \neg y) \Leftrightarrow (\neg x \downarrow \neg y)$   
 $\Leftrightarrow (x \downarrow x) \downarrow (y \downarrow y)$
- $x \vee y \Leftrightarrow \neg\neg(x \vee y) \Leftrightarrow \neg(x \downarrow y)$   
 $\Leftrightarrow (x \downarrow y) \downarrow (x \downarrow y)$

