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.

τ vs $\hat{\tau}$

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

Connectives

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

- כללי האפס והיחידה:

$X \cdot 1 \Leftrightarrow X$ $\quad$ $X + 0 \Leftrightarrow X$

$X \cdot 0 \Leftrightarrow 0$ $\quad$ $X + 1 \Leftrightarrow 1$

- כללי הכפילות:

$X \cdot X \Leftrightarrow X$ $\quad$ $X + X \Leftrightarrow X$

- כללי היפוך:

$X \cdot \overline{X} \Leftrightarrow 0$ $\quad$ $X + \overline{X} \Leftrightarrow 1$

- כלל הבליעה הראשון:

$x + xy \Leftrightarrow x$

- כלל הבליעה השני:

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

- חוקי דה-מורגן (לשני משתנים):

1. $\overline{x \cdot y} \Leftrightarrow \bar{x} + \bar{y}$
2. $\overline{x + y} \Leftrightarrow \bar{x} \cdot \bar{y}$

Example

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

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	$X + Y$	ϕ_1
0	0	0	1
1	0	1	1
0	1	1	1
1	1	1	1

X	Y	$X \cdot Y$	ϕ_2
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	1

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

□

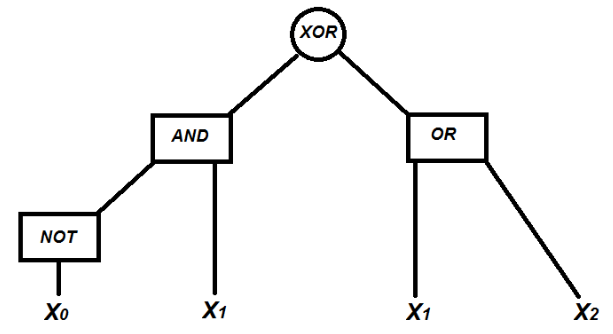
The Logisim Software

- 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.

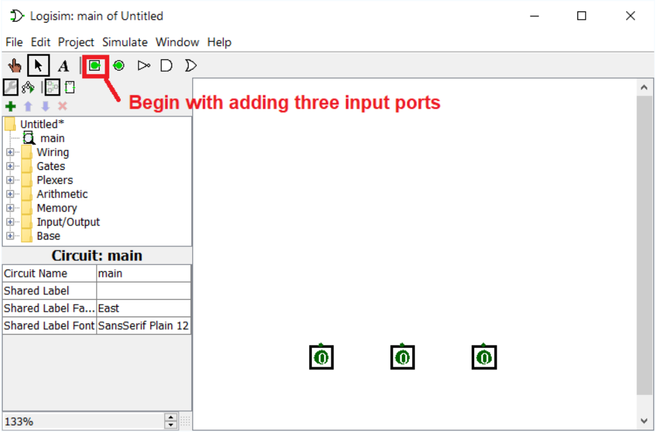
Example

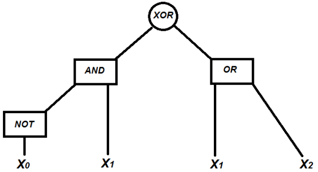
Consider the following boolean formula ϕ:
$\phi = (x_2 \vee x_1) \oplus (\neg x_0 \wedge x_1)$
Implement the boolean function $B_\phi : \{0,1\}^3 \to \{0,1\}$ that corresponds to the formula ϕ.

Parse tree of a formula ϕ

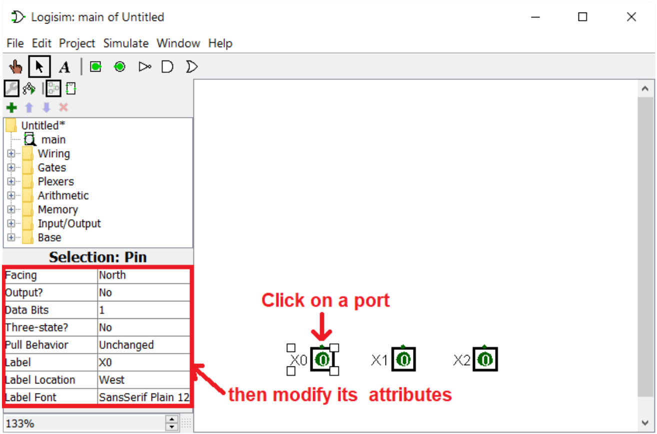


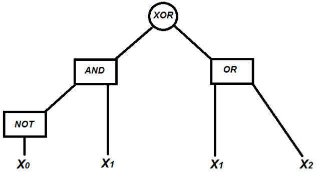
Logisim Example - B_ϕ implementation



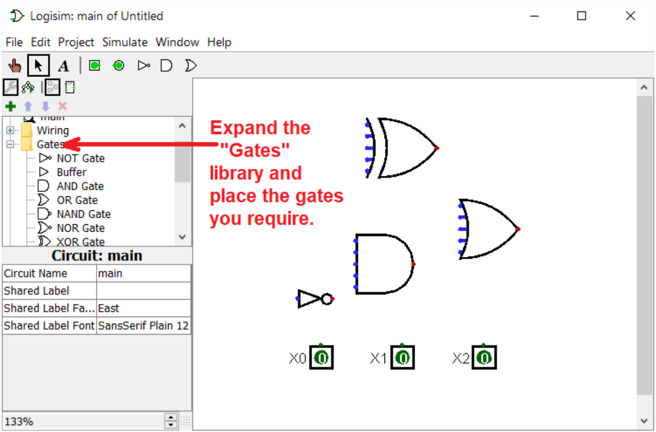


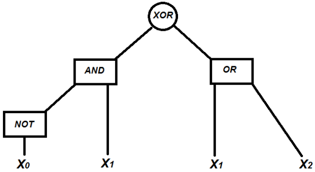
Logisim Example - B_ϕ implementation



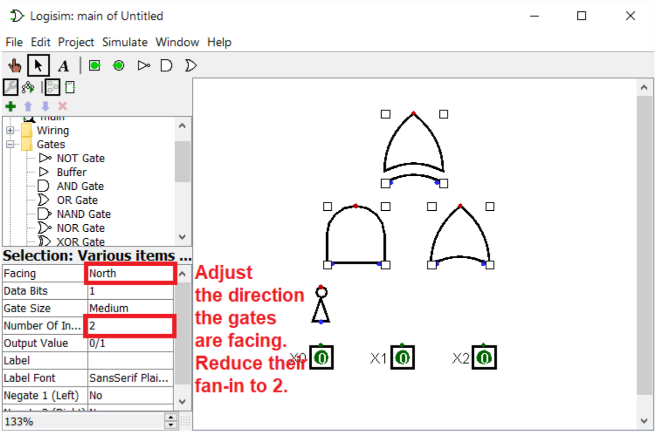


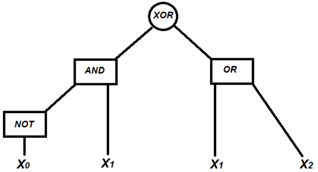
Logisim Example - B_ϕ implementation



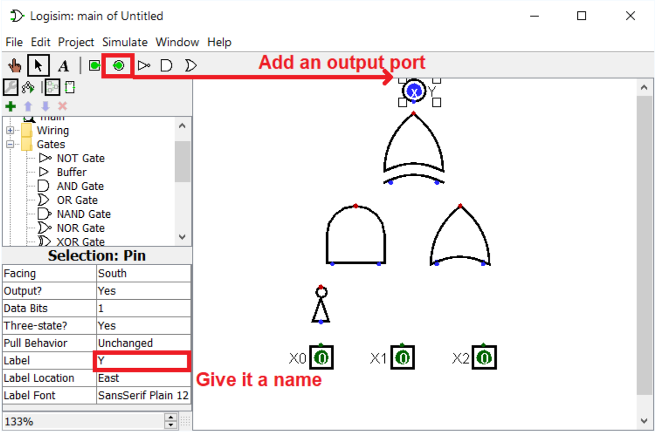


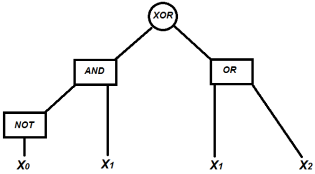
Logisim Example - B_ϕ implementation



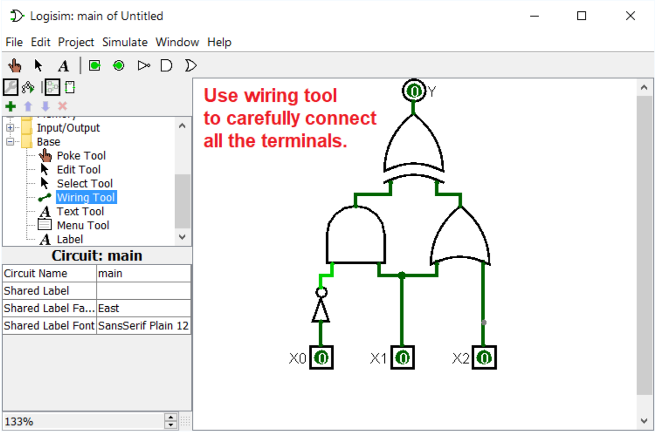


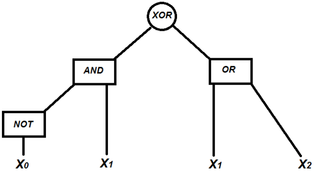
Logisim Example - B_ϕ implementation



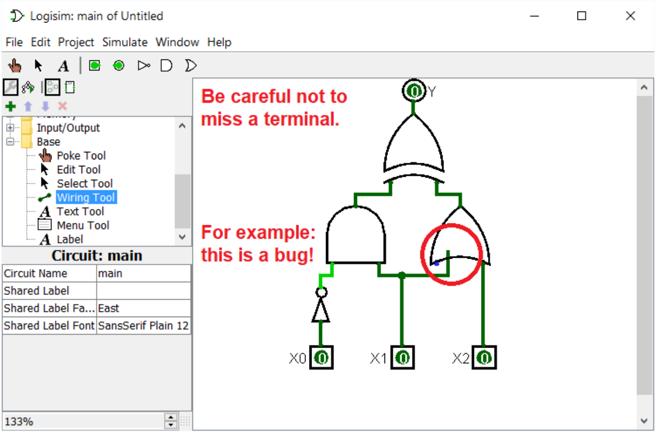


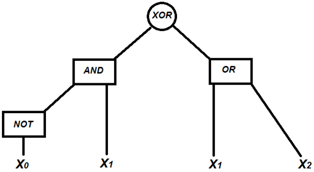
Logisim Example - B_ϕ implementation



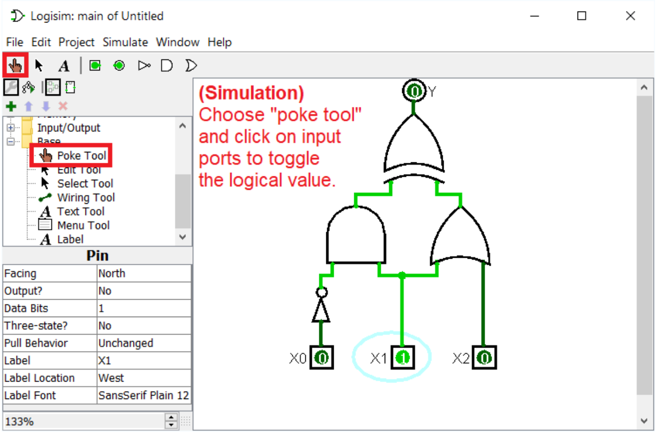


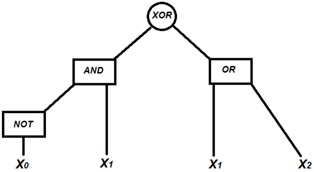
Logisim Example - B_ϕ implementation





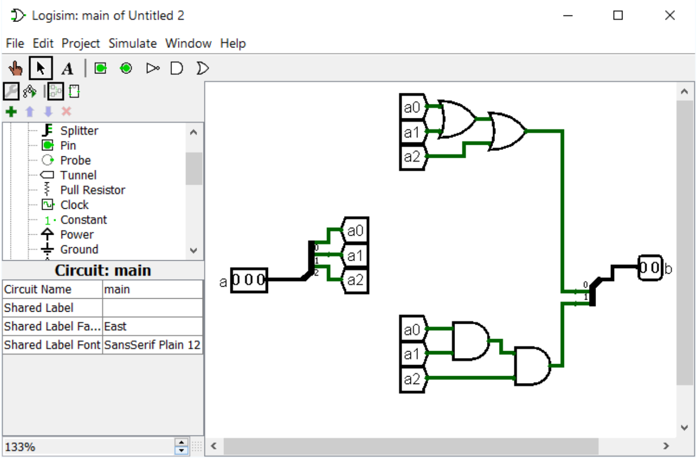
Logisim Example - B_ϕ implementation





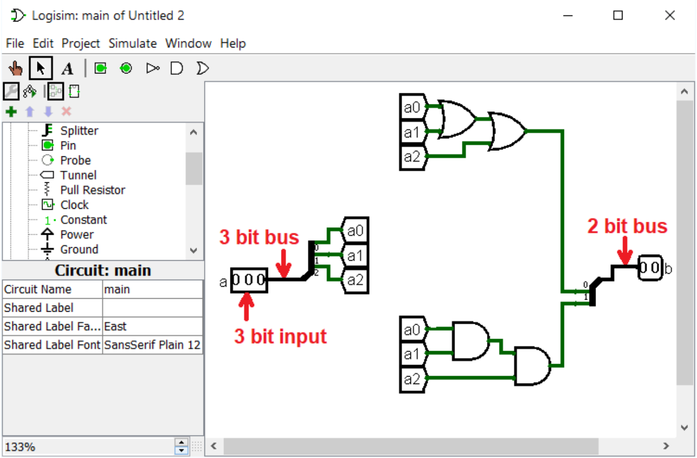
Logisim - Wiring Library

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.



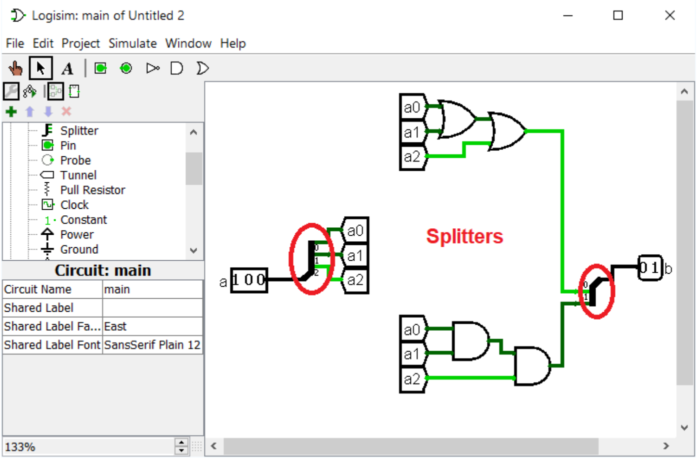
Logisim - Wiring Library

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.



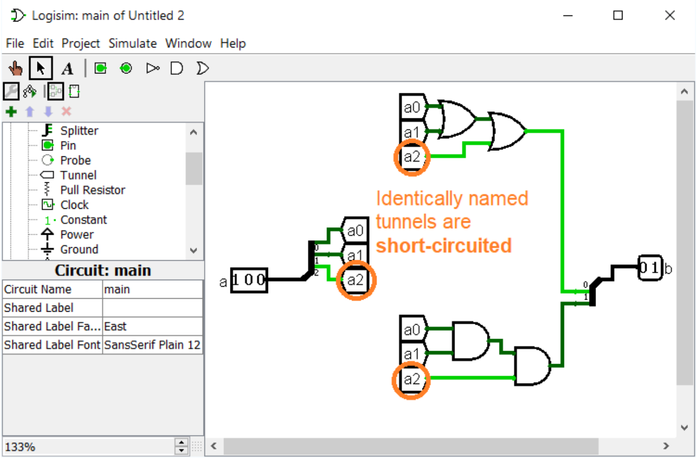
Logisim - Wiring Library

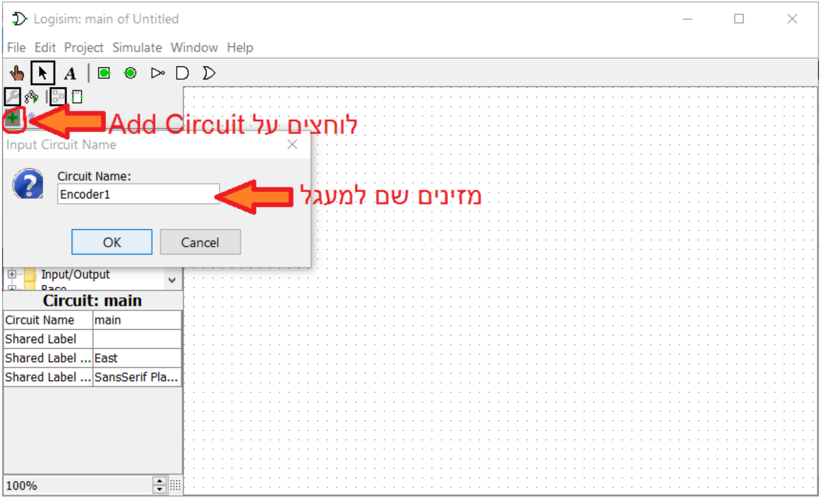
1. Bus - a set of parallel wires
2. Splitters - are used to split/collect a bus into individual wires
3. Tunnels - a.k.a. net labeling to avoid drawing connections.



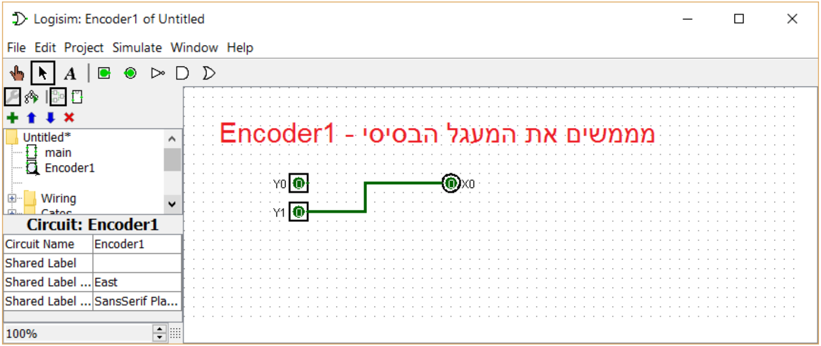
Logisim - Wiring Library

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.

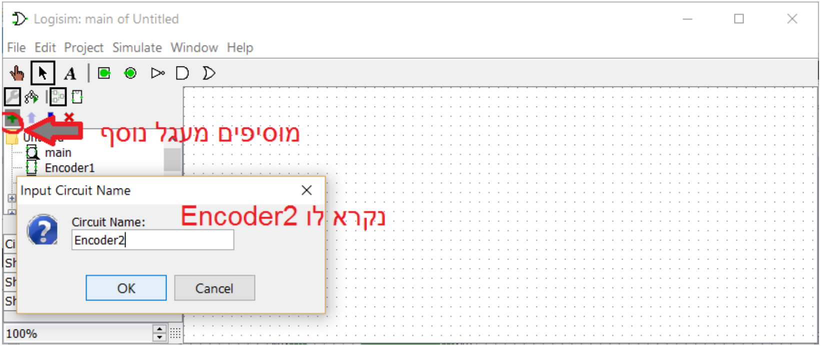


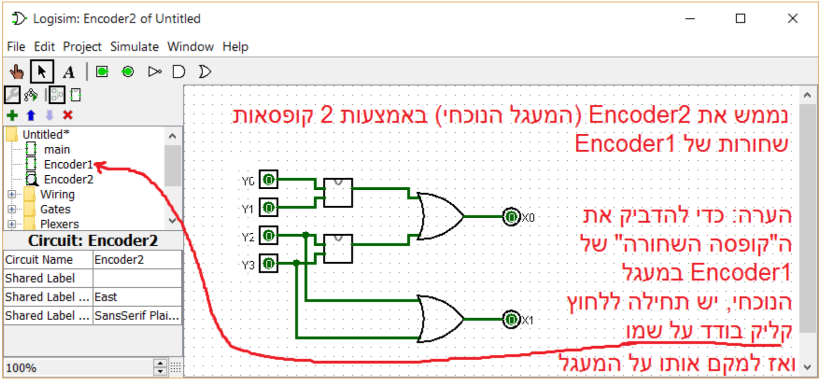

Logisim: main of Untitled
File Edit Project Simulate Window Help
Add Circuit לוחצים על
Input Circuit Name
Circuit Name:
Encoder1
מזינים שם למעגל
OK
Cancel
Input/Output
Circuit: main
Circuit Name main
Shared Label
Shared Label ... East
Shared Label ... SansSerif Pla...
100%

Modular Design in Logisim

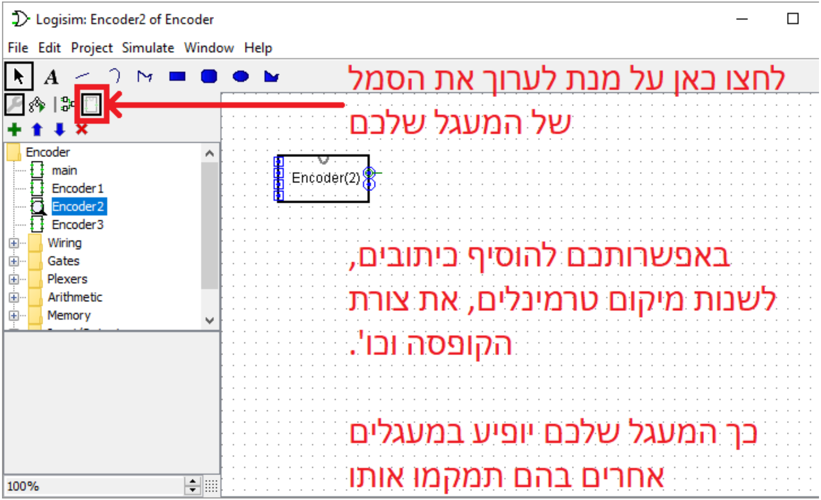


Modular Design in Logisim



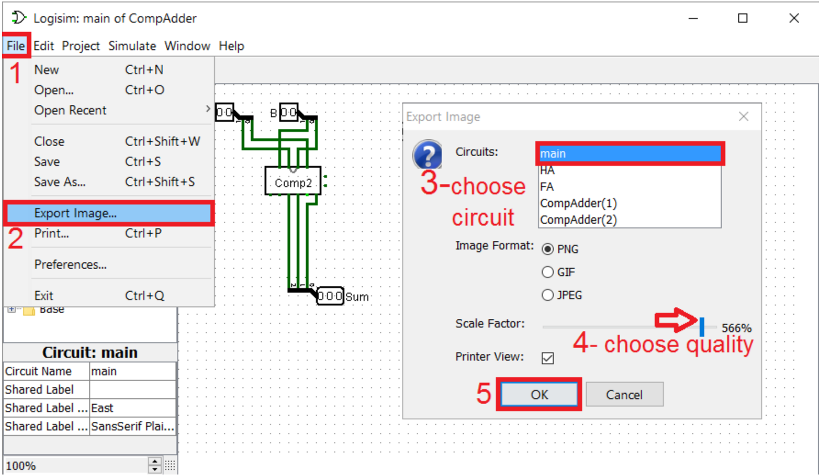

Logisim: Encoder2 of Untitled
File Edit Project Simulate Window Help
Untitled*
main
Encoder1
Encoder2
Wiring
Gates
Plexers
Circuit: Encoder2
Circuit Name Encoder2
Shared Label
Shared Label ... East
Shared Label ... SansSerif Plai...
100%
Y0
Y1
Y2
Y3
X0
X1
נממש את Encoder2 (המעגל הנוכחי) באמצעות 2 קופסאות
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

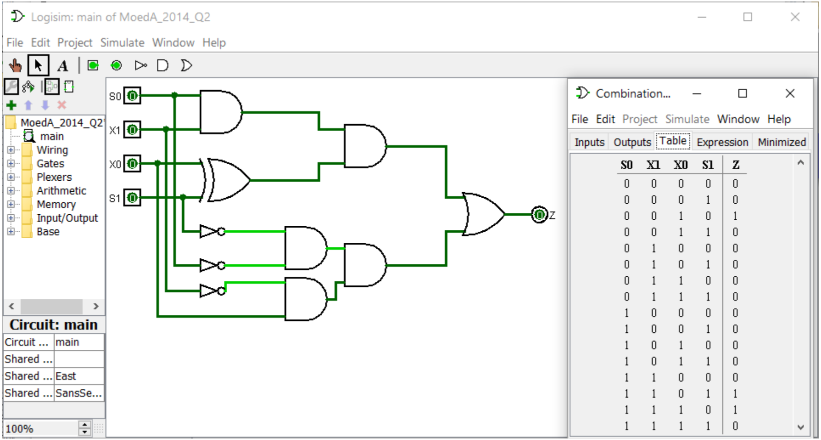


Exporting Circuit as an image without a truth table

Use the “File→Export Image” option

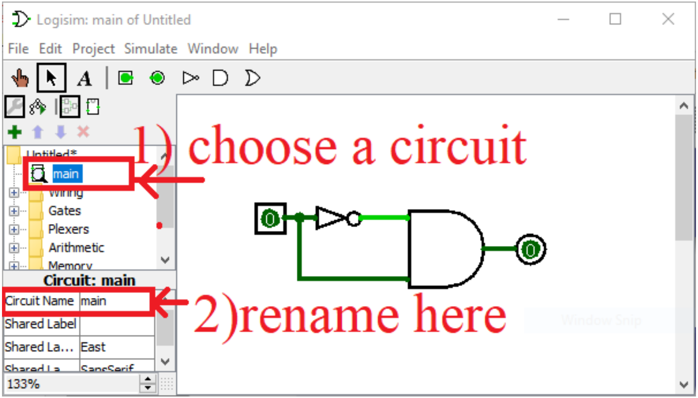


Open the “Project→Analyze Circuit”→“Table” then print screen:



Adjusting the circuit name

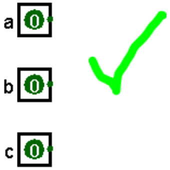
In the project submission guidelines you are required to rename your top module to a specific name.



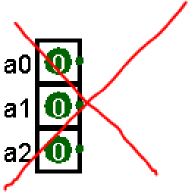
Multi-bit IO ports

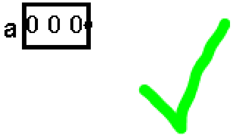
From now on, do not use multiple single-bit ports when indexing is required.

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



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





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

Definition

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

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_φ 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 φ (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 φ.

- **Base:** For trees of a size 1, φ 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 φ 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 φ.

- **Induction Step:** We shall prove that for a formula φ (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 φ 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 φ_1, φ_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_φ...

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

Table: Given B_{φ_1},B_{φ_2}, the following table describes what will be the B_φ for the both cases

B_{φ_1}	B_{φ_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 expresses 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 expresses 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 expresses 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 expresses 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)
- $x \wedge y \Leftrightarrow \neg\neg(x \wedge y) \Leftrightarrow \neg(\neg x \vee \neg y) \Leftrightarrow (\neg x \downarrow \neg y)$
  $\Leftrightarrow (x \downarrow x) \downarrow (y \downarrow y)$
- $x \vee y \Leftrightarrow \neg\neg(x \vee y) \Leftrightarrow \neg(x \downarrow y)$
  $\Leftrightarrow (x \downarrow y) \downarrow (x \downarrow y)$

□