



Computer Hardware Engineering (IS1200)

Computer Organization and Components (IS1500)

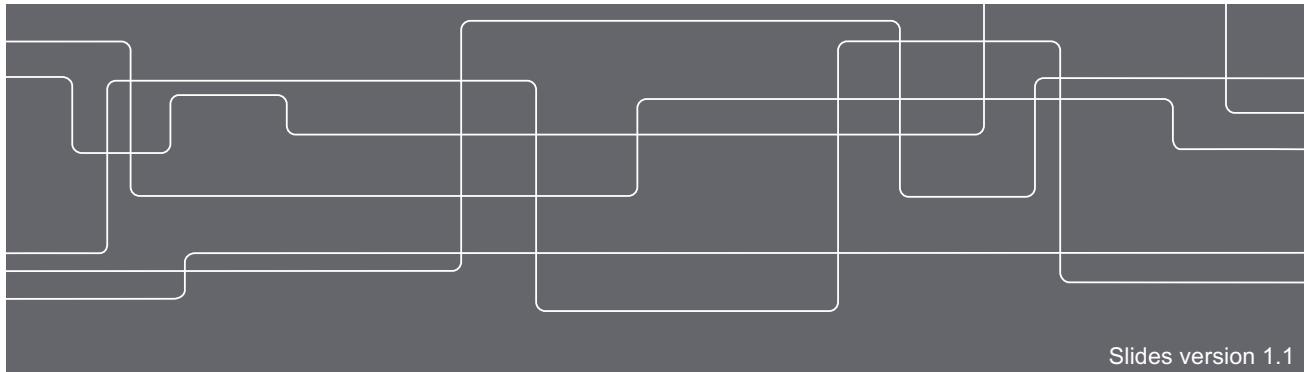
Spring 2018

Lecture 7: Combinational Logic

Note: This lecture is optional for IS1200 (for review only)

David Broman

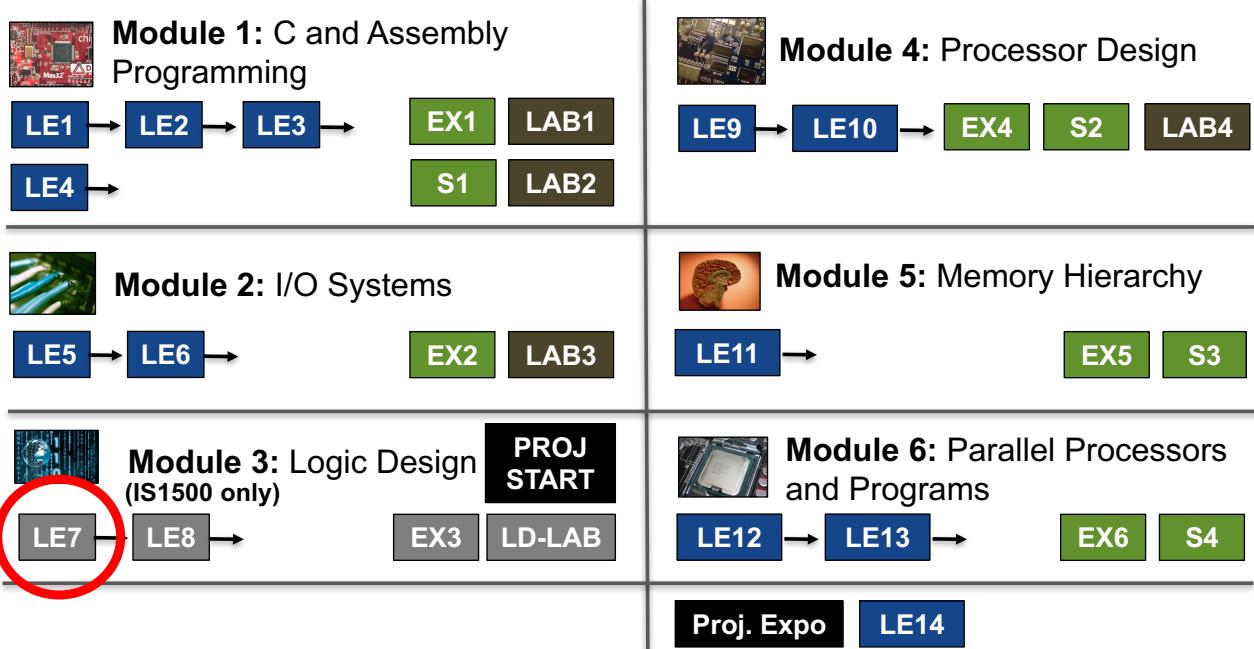
Associate Professor, KTH Royal Institute of Technology



2



Course Structure



David Broman
dbro@kth.se

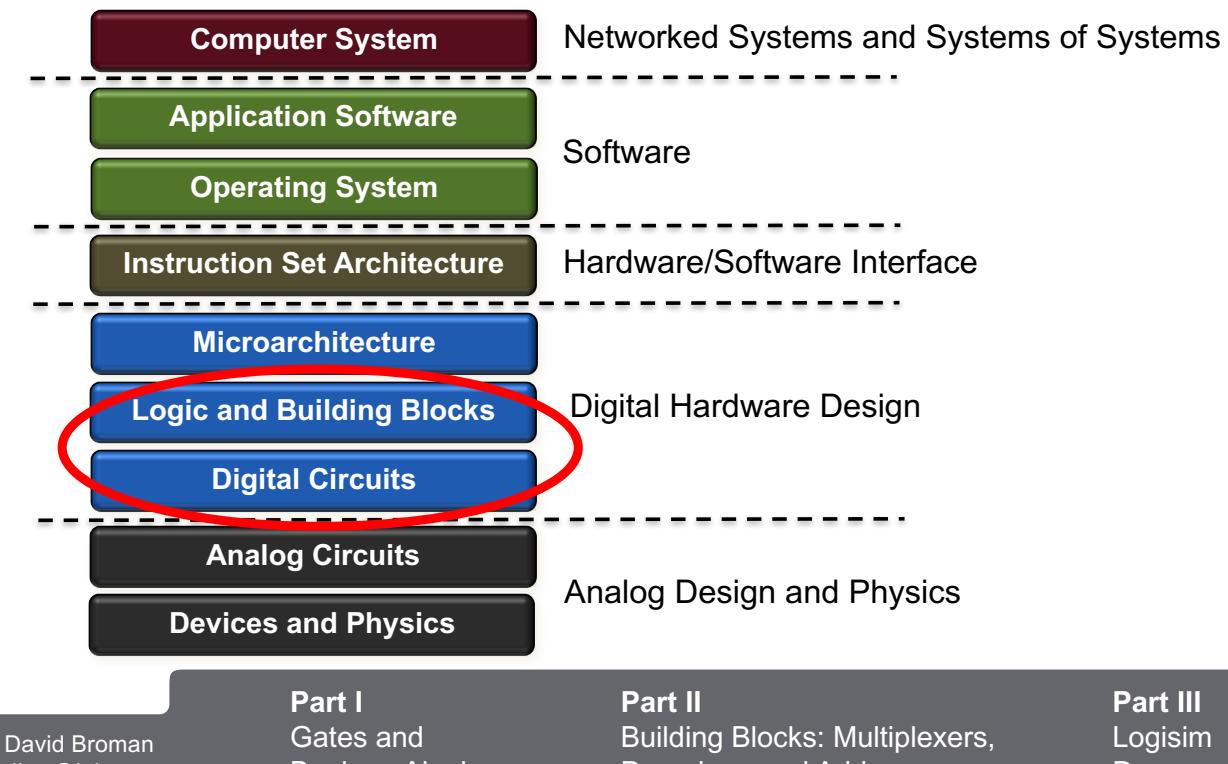
Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Abstractions in Computer Systems



David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Agenda

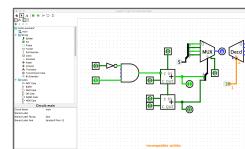
Part I Gates and Boolean Algebra



Part II Building Blocks: Multiplexers, Decoders, and Adders



Part III Logisim Demo



David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Part I

Gates and Boolean Algebra



David Broman
dbro@kth.se



Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Logic Gates (1/3) AND, OR, NOT, and BUF

AND



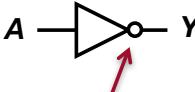
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT

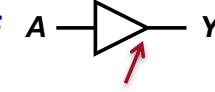


The small circle (called a *bubble*) inverse the signal.

A	Y
0	1
1	0

NOT is also called an *inverter*.

BUF



Looks like **not**, but has no circle.

A	Y
0	0
1	1

Buffer. Logically the same as a wire. Relevant from an analog point of view.

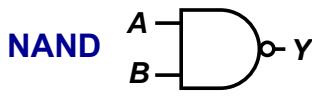
David Broman
dbro@kth.se



Part I
Gates and
Boolean Algebra

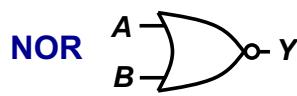
Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



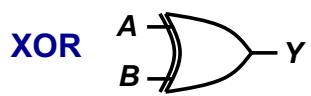
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOT AND. Note the Small bubble at the end.



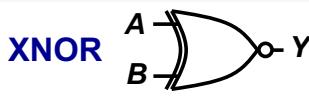
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

NOT OR.



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive OR,
pronounced "ex-or".



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

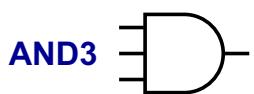
Exclusive NOT OR.



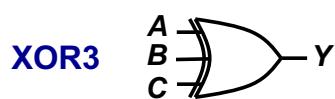
Logic Gates (3/3)

Multi-Input Logic Gates

Gates can be generalized to have more than two inputs. For instance:



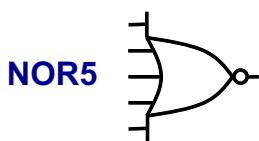
AND gate with 3 inputs.



Exclusive OR gate with 3 inputs.

An N-input XOR gate is also called a **parity gate**. It outputs 1 when odd number of inputs are 1.

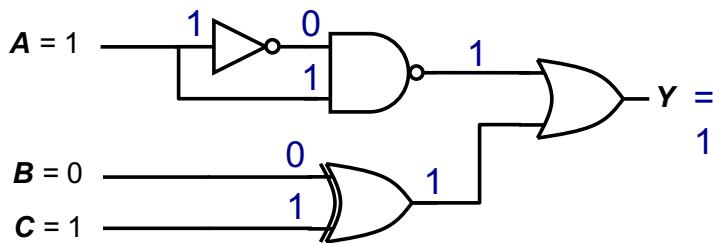
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



NOT OR gate with 5 inputs.



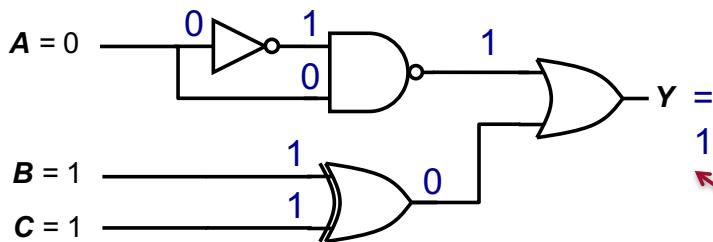
Combinational Circuit



This circuit is **combinational** because its outputs depend *only* on its inputs. The circuit is *memoryless*, that is, it has no memory.



We will introduce memory in Lecture 8



Observe that this (rather useless) circuit always outputs 1. As a logic formula, this is called a **tautology**.

David Broman
dbro@kth.se



Part I
Gates and
Boolean Algebra

Part II

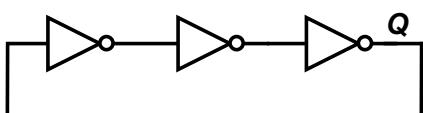
Building Blocks: Multiplexers,
Decoders, and Adders

Part III

Logisim
Demo

Problematic Circuits

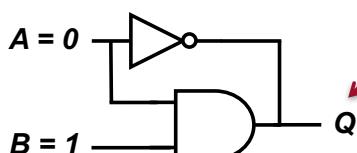
Unstable circuit.



What is the value of Q?

Answer: it oscillates. This circuit is called a **ring oscillator**.

Illegal value (X)



What is the value of Q?

Answer: Q = X, called an unknown or illegal value. For example, when a wire is driven to both 0 and 1 at the same time.

This situation is called **contention**.

David Broman
dbro@kth.se



Part I
Gates and
Boolean Algebra

Part II

Building Blocks: Multiplexers,
Decoders, and Adders

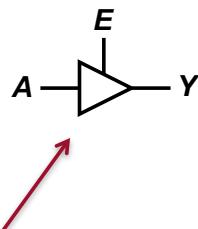
Part III

Logisim
Demo



Floating Values and Tristate Buffers

A tristate (or three-state) buffer has high impedance if the output enable signal E is not active.



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

Commonly used in **buses** when connecting multiple chips. If the buffers are not enabled at the same time, contention is avoided.

When the enable signal is not active, the output is said to be **floating** (using symbol Z).

David Broman
dbro@kth.se



Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Boolean Algebra (1/4) Truth Tables and Sum-of-Products Form

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A truth table with random output (we have seen them before).

We can create a boolean expression from the truth table

The **AND** of one or more variables is called a **product**.

$\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}\bar{C}$
AND can be written using “no space” or using a dot, e.g. $A \cdot B \cdot C$

The line over a variable is called the **complement** and is the inverse of the variable (NOT). Sometimes a prime ‘ is used instead.

OR is written using the + symbol.

This form is called **sum-of-products** (surprise!)

David Broman
dbro@kth.se



Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Theorem	Dual	Name	
$A \cdot 1 = A$	$A + 0 = A$	Identity	Exercise: Derive the simplest form of expression $BA + A\bar{B} + A$
$A \cdot 0 = 0$	$A + 1 = 1$	Null Element	
$A \cdot A = A$	$A + A = A$	Idempotency	
$\bar{\bar{A}} = A$		Involution	
$A \cdot \bar{A} = 0$	$A + \bar{A} = 1$	Complements	
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutativity	
$(A \cdot B) \cdot C =$ $A \cdot (B \cdot C)$	$(A + B) + C =$ $A + (B + C)$	Associativity	
$(A \cdot B) + (A \cdot C) =$ $A \cdot (B + C)$	$(A + B) \cdot (A + C) =$ $A + (B \cdot C)$	Distributivity	

Note! Not as traditional algebra

David Broman
dbro@kth.se



Part I

Gates and Boolean Algebra

Part II

Building Blocks: Multiplexers, Decoders, and Adders

Part III Logisim Demo



Boolean Algebra (3/4)

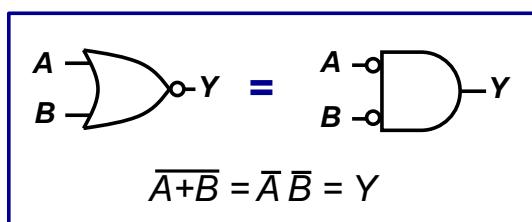
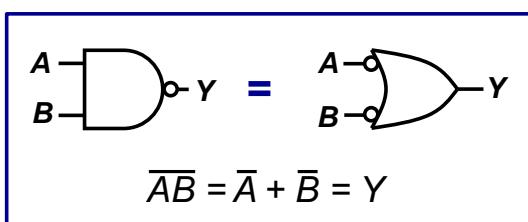
De Morgan's Theorem

Theorem	Dual
$\overline{A_1 \cdot A_2 \cdot A_3 \dots} = (\overline{A}_1 + \overline{A}_2 + \overline{A}_3 \dots)$	$\overline{A_1 + A_2 + A_3 \dots} = (\overline{A}_1 \cdot \overline{A}_2 \cdot \overline{A}_3 \dots)$



Augustus De Morgan
British mathematician and
logician (1806 – 1871).

The law shows that these gates are equivalent



Important law. For CMOS (Complementary metal–oxide–semiconductor), **NAND** and **NOR** gates are preferred over AND and OR gates.

But how can we know that this theorem is true?

David Broman
dbro@kth.se



Part I Gates and Boolean Algebra

Part II

Building Blocks: Multiplexers, Decoders, and Adders

Part III
Logisim
Demo

Perfect Induction = Proof by Exhaustion = Proof by Cases

Prove the De Morgan's Theorem for three variables

$$\overline{ABC} = \overline{A} + \overline{B} + \overline{C}$$

Proof by **perfect induction**.
Exhaustively show all cases
in a truth table.

Note that these two columns are equal

A	B	C	\overline{ABC}	$\overline{A} + \overline{B} + \overline{C}$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

David Broman
dbro@kth.se



Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Part II

Building Blocks: Multiplexers, Decoders, and Adders



David Broman
dbro@kth.se

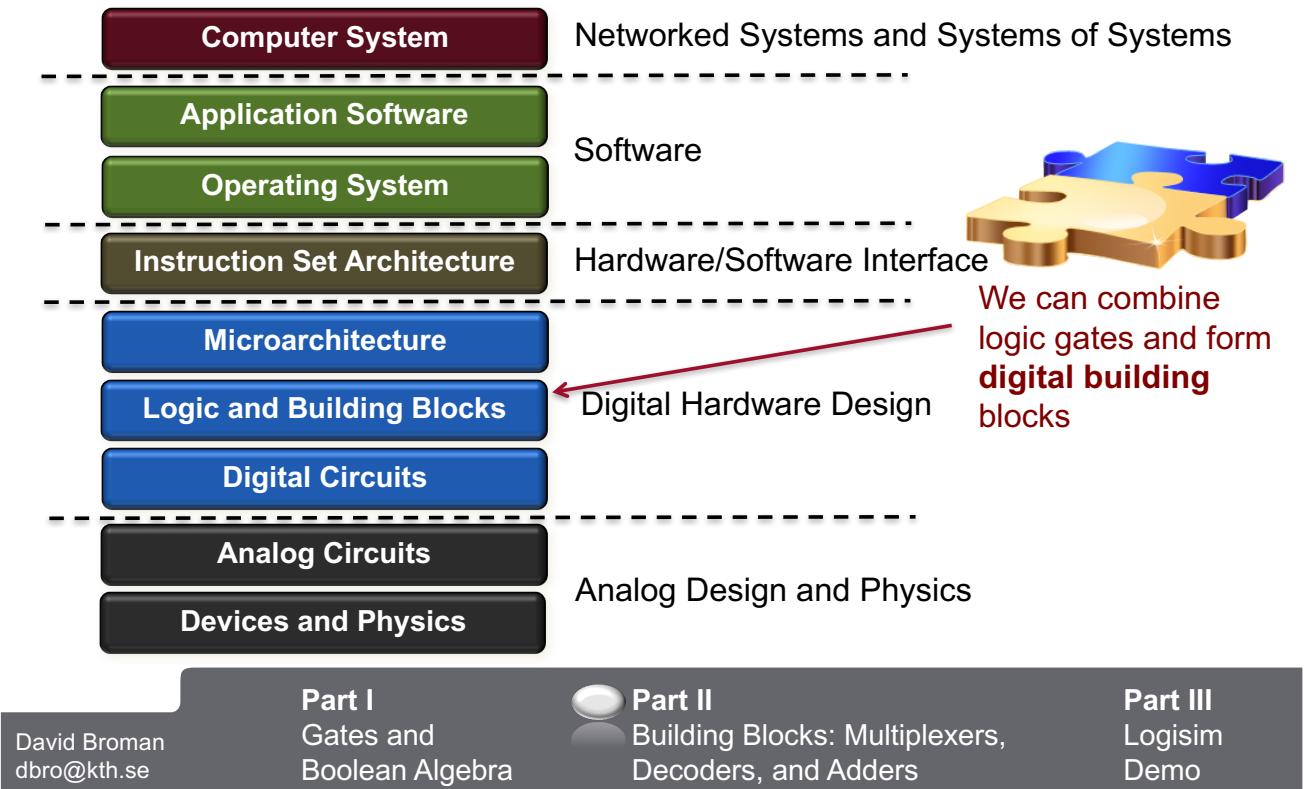
Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Abstractions in Computer Systems



Combinational Blocks (1/3) Multiplexers

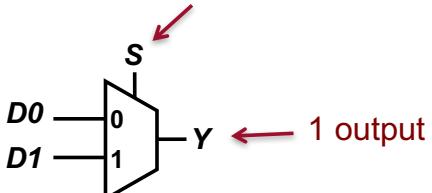


What is this?

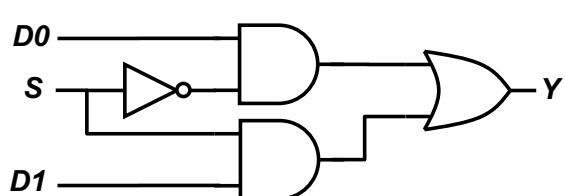
S	D1	D0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

It's a **2:1 Multiplexer**.

2 bits for the
data input



The control signal **S** selects which input bit that is sent to the output.

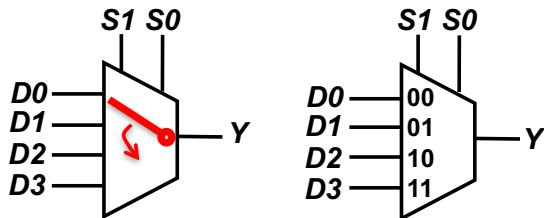


One possible
implementation.
Convince
yourself of its
correctness!



Combinational Blocks (2/3) Multiplexers

A **multiplexer** can be seen as a simple switch, selecting which signal that should pass through the block.

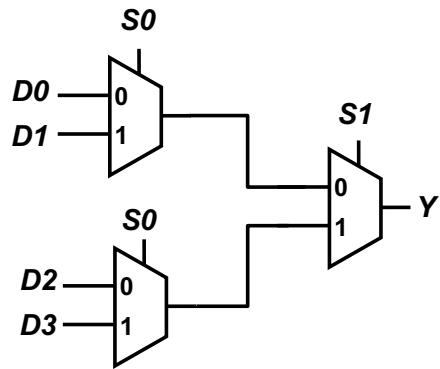


4:1 multiplexer (4 inputs, 1 output).

What is the output signal Y for the 4:1 multiplexer with these inputs?

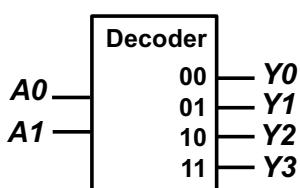
$D0 = 1, D1 = 0, D2=1, D3=0,$
 $S1 = 1, S0 = 0$ Answer: $Y = 1$

A 4:1 multiplexer can be defined hierarchically.



Combinational Blocks (3/3) Decoders

A **decoder** has N inputs and 2^N outputs.
Asserts exactly one output.

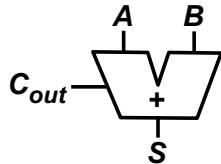


2:4 decoder (2 inputs, 4 output).

A1	A0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Note that only one signal is 1 on each row. This is called **one-hot**.

A **half adder** has a *carry out* signal.



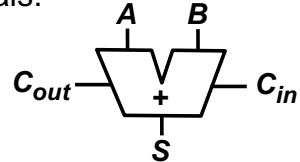
How can we add bigger numbers?

Idea: Chain adders together...



A	B	S	C _{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A **full adder** has both *carry out* and *carry in* signals.



C _{in}	A	B	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Exercise:
Complete the truth table

David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

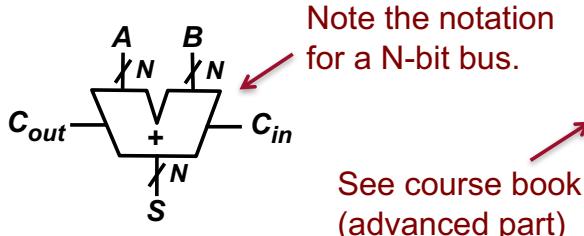
Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo

Arithmetic Circuits and Numbers (2/7)

Carry Propagate Adders

An N-bit **carry propagate adder (CPA)** sums two N-bit inputs.

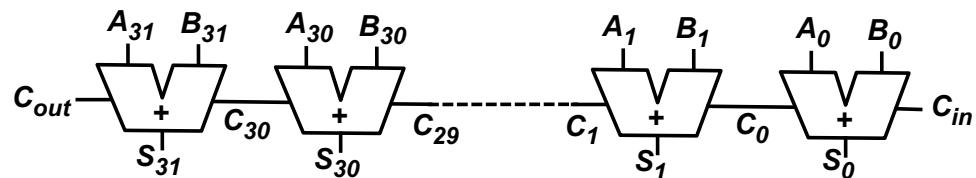


Three common implementations of CPAs are:

- **Ripple-carry adder**
Simple but slow.
- **Carry-lookahead adder**
Faster, divides into blocks.
- **Prefix adder**
Even faster. Used in modern computers.



32-bit ripple-carry adder



David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

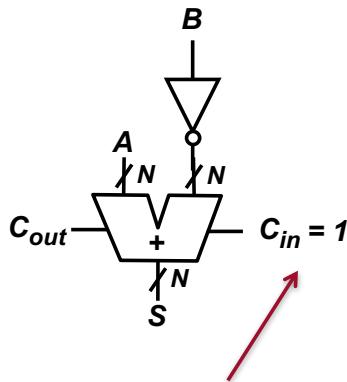
Part III
Logisim
Demo



Arithmetic Circuits and Numbers (7/7)

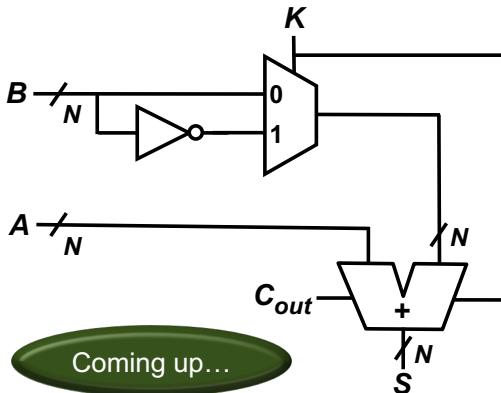
Subtract

Subtract is simple to implement with a **carry propagate adder (CPA)**:
Invert input signal B and set $C_{in} = 1$.



Note that setting **carry in** to 1 adds 1 to $A + B$.

We can easily create a circuit where $K = 0$ results in $A + B$ and $K = 1$ results in $A - B$



Coming up...

In lecture 9, we will generalize this idea into an **Arithmetic/Logic Unit (ALU)**, one of the main components of a processor.

David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

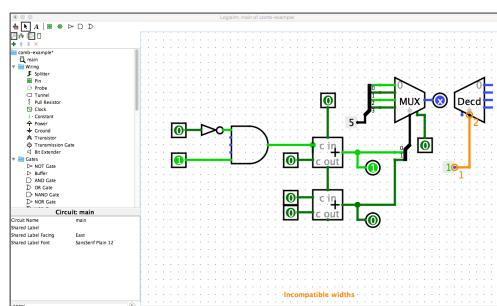
Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Part III

Logisim Demo



David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo

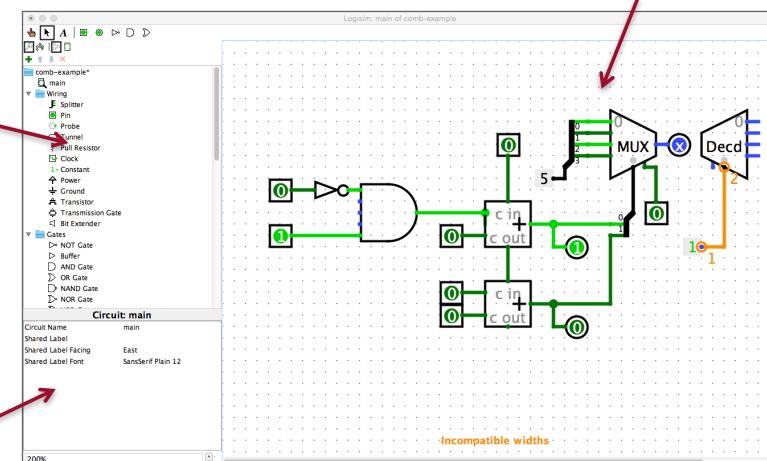


Logisim

Free graphical digital circuit simulator.
Used in the LD-LAB and in LAB4.

Graphical Model Canvas
Both for construction and simulation

Explorer Pane
Building blocks
and gates



Attribute Table
Configure
different
components

David Broman
dbro@kth.se

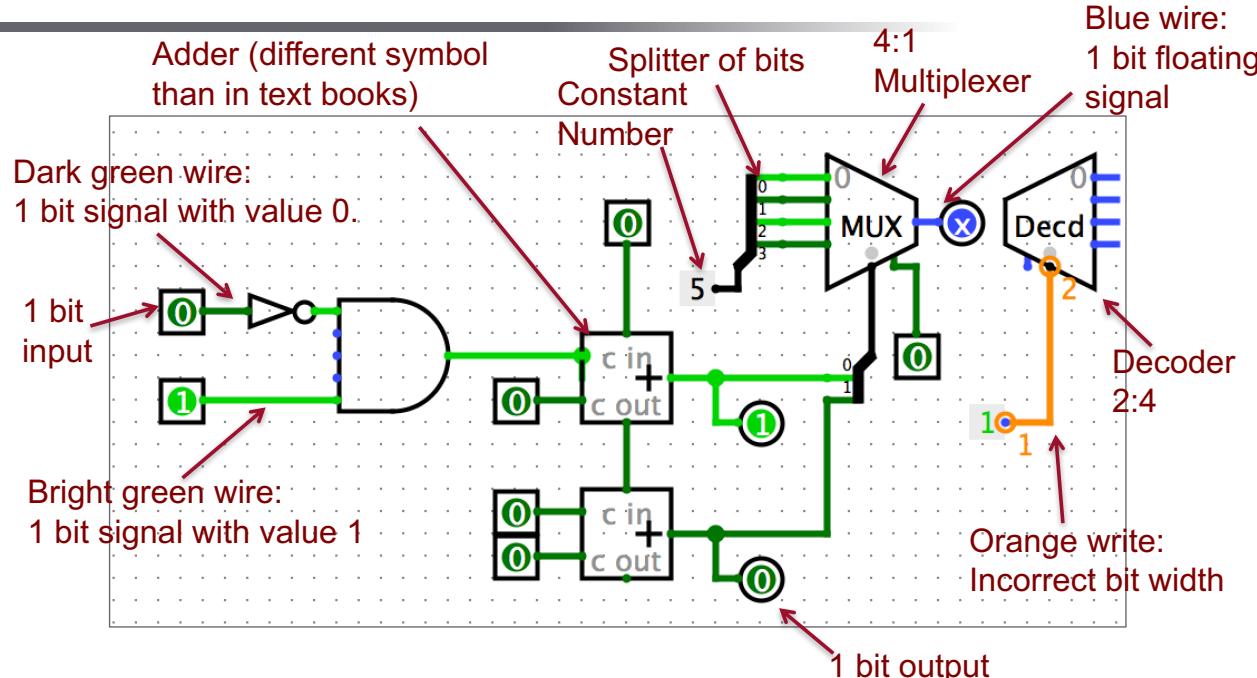
Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

 **Part III**
Logisim
Demo



Some Different Notations in Logisim



David Broman
dbro@kth.se

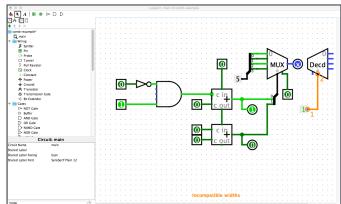
Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

 **Part III**
Logisim
Demo



Hardware Description Languages



Logisim is a simple graphical design and simulation environment for *educational purposes*.

For those who are interested, see Harris & Harris (2012), chapter 4. This is not part of the course.

Professional hardware designers work in textual Hardware Description Languages (HDL).

The two most commonly used HDLs in industry are:

- **System Verilog**. Used a lot in USA. C-like syntax.
- **VHDL**. Used more in Europe. Ada-like syntax.

It also exists recent domain-specific languages (DSLs) for hardware design. Ex. Chisel from UC Berkeley (embedded in Scala).

David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

 **Part III**
Logisim
Demo



Please...
...do not fumble with the bags



David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Reading Guidelines



Module 3: Logic Design

Lecture 7: Combinational Logic Design

- H&H Chapters 1.5, 2.1-2.4, 2.6, 2.8-2.9

Lecture 8: Sequential Logic Design

- H&H Chapters 3.1-3.3 (not 3.2.7),
3.4.1-3.4.3, 5.2.1-5.2.2, 5.5.5

Reading Guidelines

See the course webpage
for more information.

David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo



Summary

Some key take away points:

- **Combinational logic design:** Output is directly dependent on input. There is no memory.
- Main components to remember: **multiplexer, decoder, and adder.**
- **Next lecture** is about **sequential logic design;** circuits with memory.



Thanks for listening!

David Broman
dbro@kth.se

Part I
Gates and
Boolean Algebra

Part II
Building Blocks: Multiplexers,
Decoders, and Adders

Part III
Logisim
Demo