Computer Hardware Engineering (IS1200)
Computer Organization and Components (IS1500)

Fall 2020

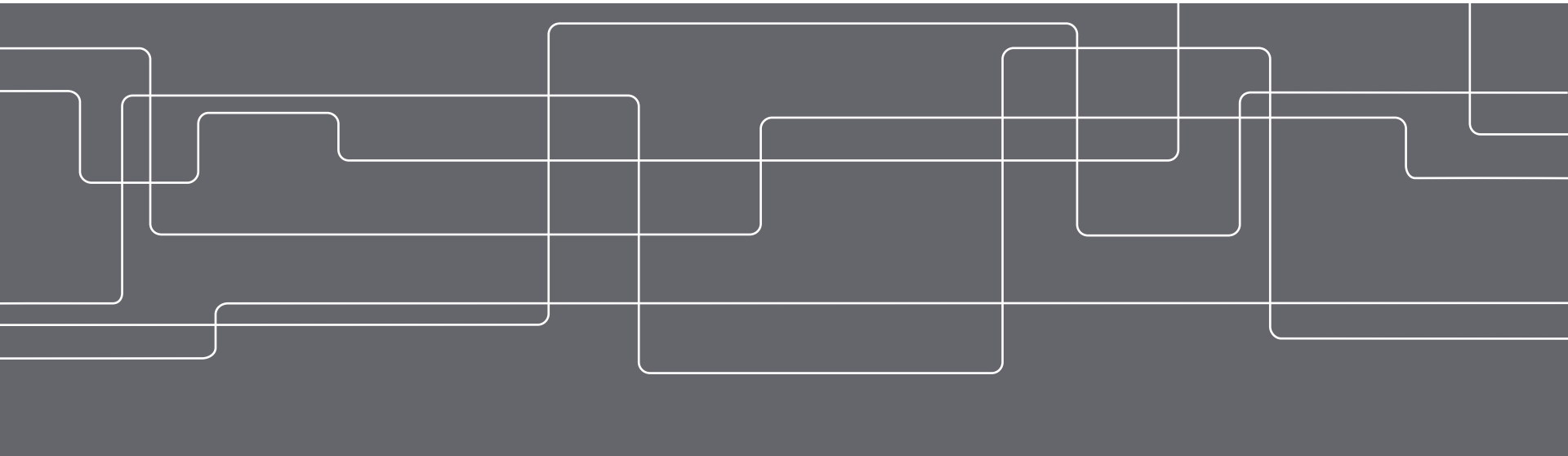
Lecture 7: Combinational Logic

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

Artur Podobas

Researcher, KTH Royal Institute of Technology

Slides by David Broman (extensions by Artur Podobas), KTH



Course Structure

Module 1: C and Assembly Programming

LE1 → LE2 → LE3 → LE4 →

EX1 LAB1 S1 LAB2

Module 2: I/O Systems

LE5 → LE6 →

EX2 LAB3

Module 3: Logic Design **(IS1500 only)**

PROJ START

LE7 → LE8 →

EX3 LD-LAB

Module 4: Processor Design

LE9 → LE10 → EX4 S2 LAB4

Module 5: Memory Hierarchy

LE11 → EX5 S3

Module 6: Parallel Processors and Programs

LE12 → LE13 → EX6 S4

Proj. Expo LE14

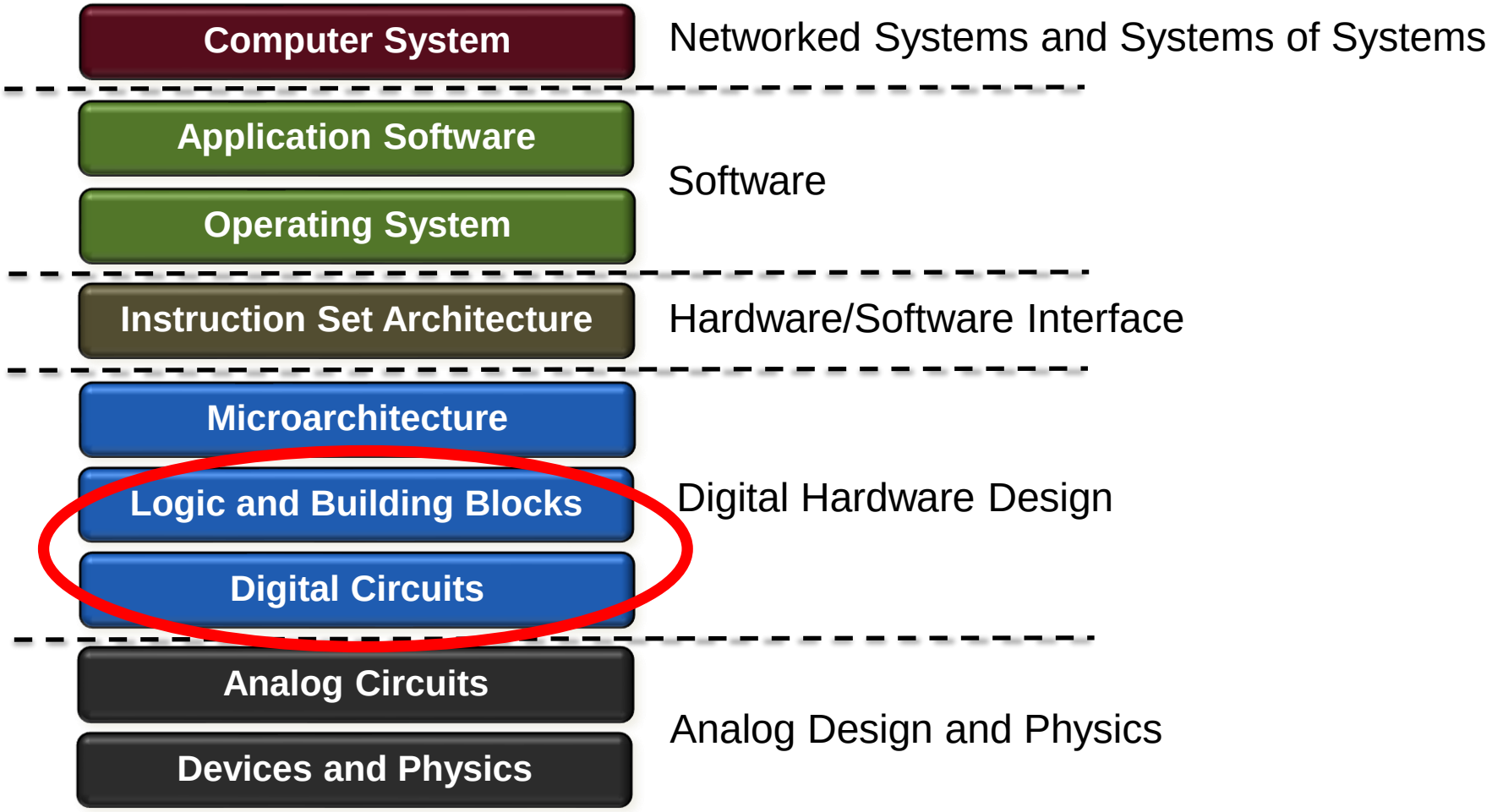
Artur Podobas
podobas@kth.se

Part I
Gates and Boolean Algebra

Part II
Building Blocks: Multiplexers, Decoders, and Adders

Part III
Logisim Demo

Abstractions in Computer Systems



Agenda

Part I

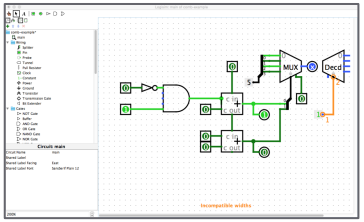
Gates and Boolean Algebra

Part II

Building Blocks: Multiplexers, Decoders, and Adders

Part III

Logisim Demo



Part I

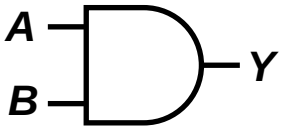
Gates and Boolean Algebra

Acknowledgement: The structure and several of the good examples are derived from the book “Digital Design and Computer Architecture” (2013) by D. M. Harris and S. L. Harris.

Artur Podobas
podobas@kth.se

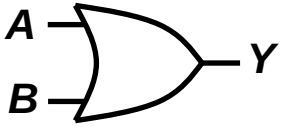
Logic Gates (1/3)
AND, OR, NOT, and BUF

AND



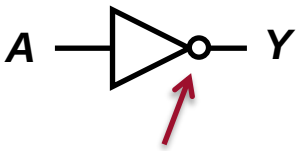
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR



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

NOT

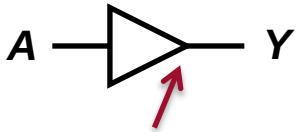


The small circle (called a *bubble*) inverses the signal.

A	Y
0	1
1	0

NOT is also called an *inverter.*

BUF



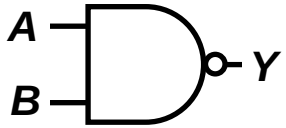
Looks like **not**, but has no circle.

A	Y
0	0
1	1

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

Logic Gates (2/3)
NAND, NOR, XOR, and XNOR

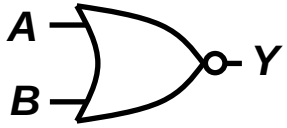
NAND



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

NOT AND. Note the Small bubble at the end.

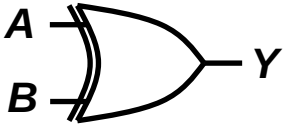
NOR



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

NOT OR.

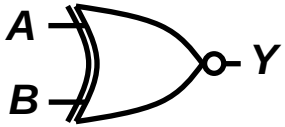
XOR



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

Exclusive OR, pronounced “ex-or”.

XNOR



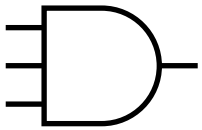
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Exclusive NOT OR.

Logic Gates (3/3)
Multi-Input Logic Gates

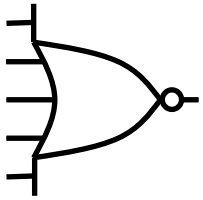
Gates can be generalized to have more than two inputs. For instance:

AND3



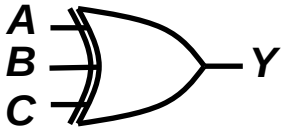
AND gate with 3 inputs.

NOR5



NOT OR gate with 5 inputs.

XOR3

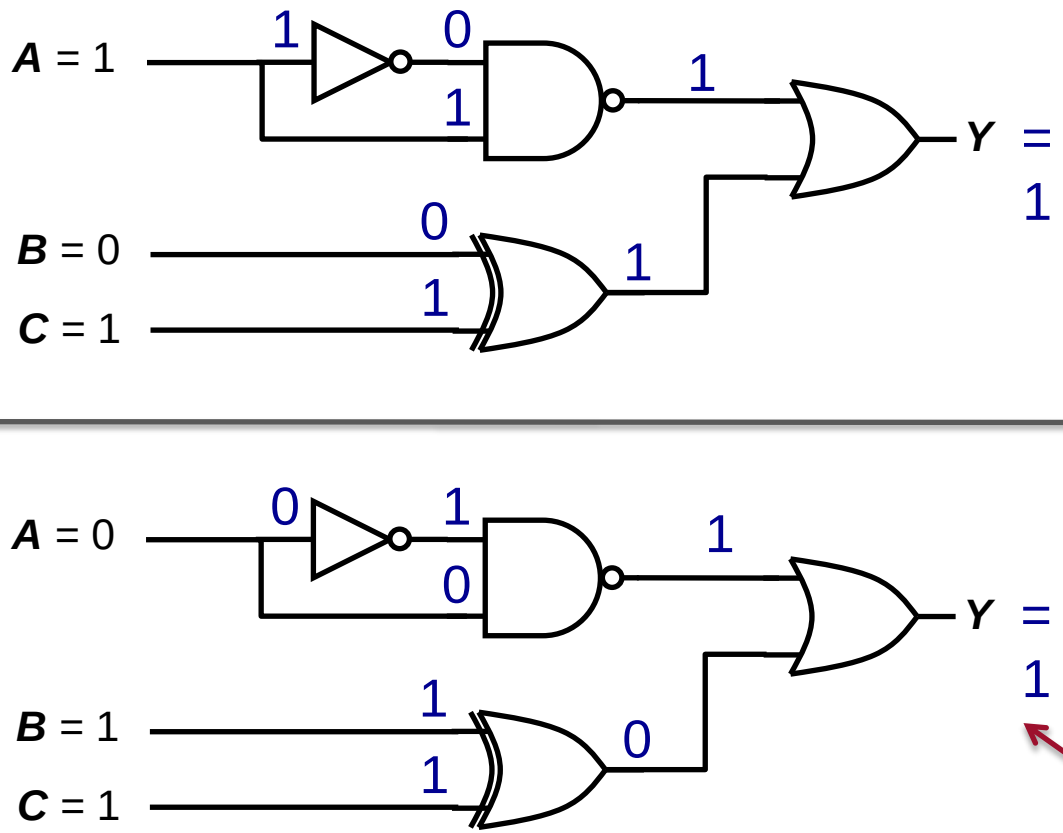


Exclusive OR gate with 3 inputs.

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	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Combinational Circuit

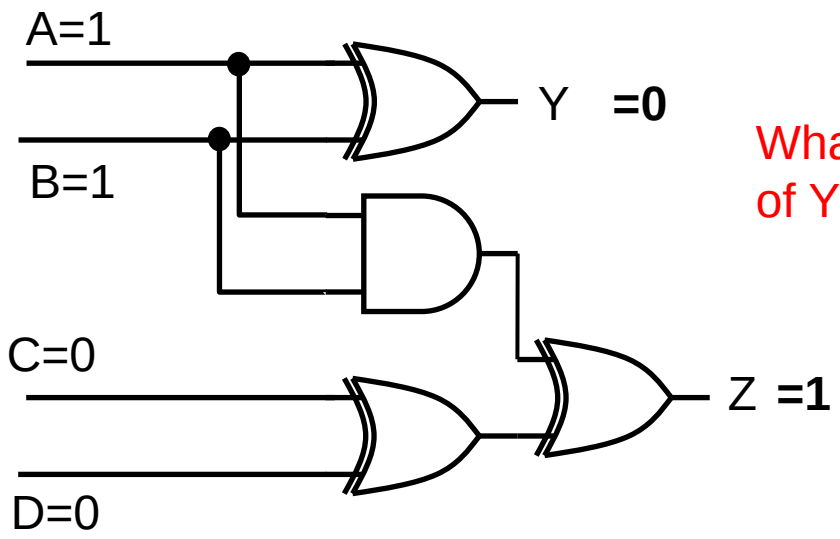


This circuit is **combinational** because its outputs depend *only* on its inputs. The circuit is *memoryless*, that is, it has no memory.

We will introduce memories in Lecture 8

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

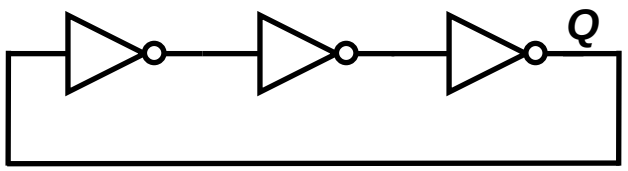
Combinational Circuit



What is the result of Y and Z?

Problematic Circuits

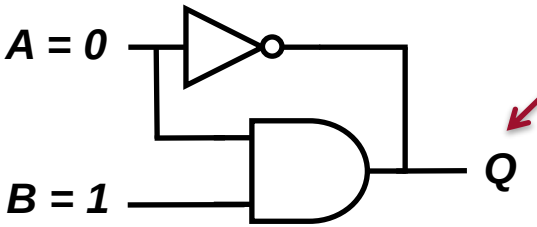
Unstable circuit.



What is the value of Q?

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

Illegal value (X)



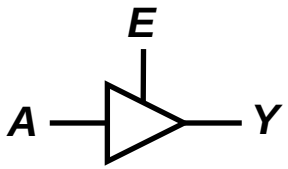
What is the value of Q?

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**.

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).

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

$$\bar{A}B\bar{C} + A\bar{B}C + AB\bar{C}$$

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

AND can be written using "no space" or using a dot, e.g. A•B•C

OR is written using the **+** symbol.

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

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

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

b_2	b_1	b_0	c_1	c_0
0	0	0	1	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Count leading zeroes: Given a bit vector $b_2b_1b_1$, count the number of leading zeroes and output as c_1c_0.

$$Y = \overline{ABC} + \overline{ABC}$$

$$Z = \overline{ABC} + \bar{A}B\bar{C} + \bar{A}BC$$

Boolean Algebra (2/4)
Some Theorems

Theorem	Dual	Name
$A \bullet 1 = A$	$A+0 = A$	Identity
$A \bullet 0 = 0$	$A+1 = 1$	Null Element
$A \bullet A = A$	$A+A = A$	Idempotency
$\overline{\overline{A}} = A$		Involution
$A \bullet \overline{A} = 0$	$A+\overline{A} = 1$	Complements
$A \bullet B = B \bullet A$	$A+B = B+A$	Commutativity
$(A \bullet B) \bullet C = A \bullet (B \bullet C)$	$(A+B)+C = A+(B+C)$	Associativity
$(A \bullet B)+(A \bullet C) = A \bullet (B+C)$	$(A+B) \bullet (A+C) = A+(B \bullet C)$	Distributivity

Note! Not as traditional algebra

Exercise:
Derive the simplest form of expression

$BA + A\overline{B} + A$

Solution:

$BA + A\overline{B} + A$

$= AB + A\overline{B} + A$ **Commutativity**

$= A(B+\overline{B}) + A$ **Distributivity**

$= A \bullet 1 + A$ **Complements (dual)**

$= A + A$ **Identity**

$= A$ **Indempotency (dual)**

Boolean Algebra (3/4)
De Morgan's Theorem

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

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

The law shows that these gates are equivalent

$$\overline{AB} = \overline{A} + \overline{B} = Y$$

$$\overline{A+B} = \overline{A}\,\overline{B} = Y$$

Important law. For CMOS (Complementary metal–oxide–semiconductor), **NAND** and **NOR** gates are preferred over AND and OR gates, and are also universal (any other gates can be constructed using e.g., NAND).

But how can we know that this theorem is true?

Boolean Algebra (4/4)
Proof by Perfect Induction

Perfect Induction = Proof by Exhaustion = Proof by Cases

Prove the De Morgan's Theorem for three variables

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

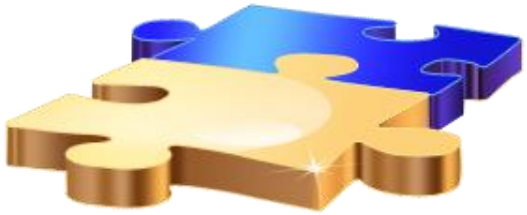
Note that these two columns are equal

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

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

Part II

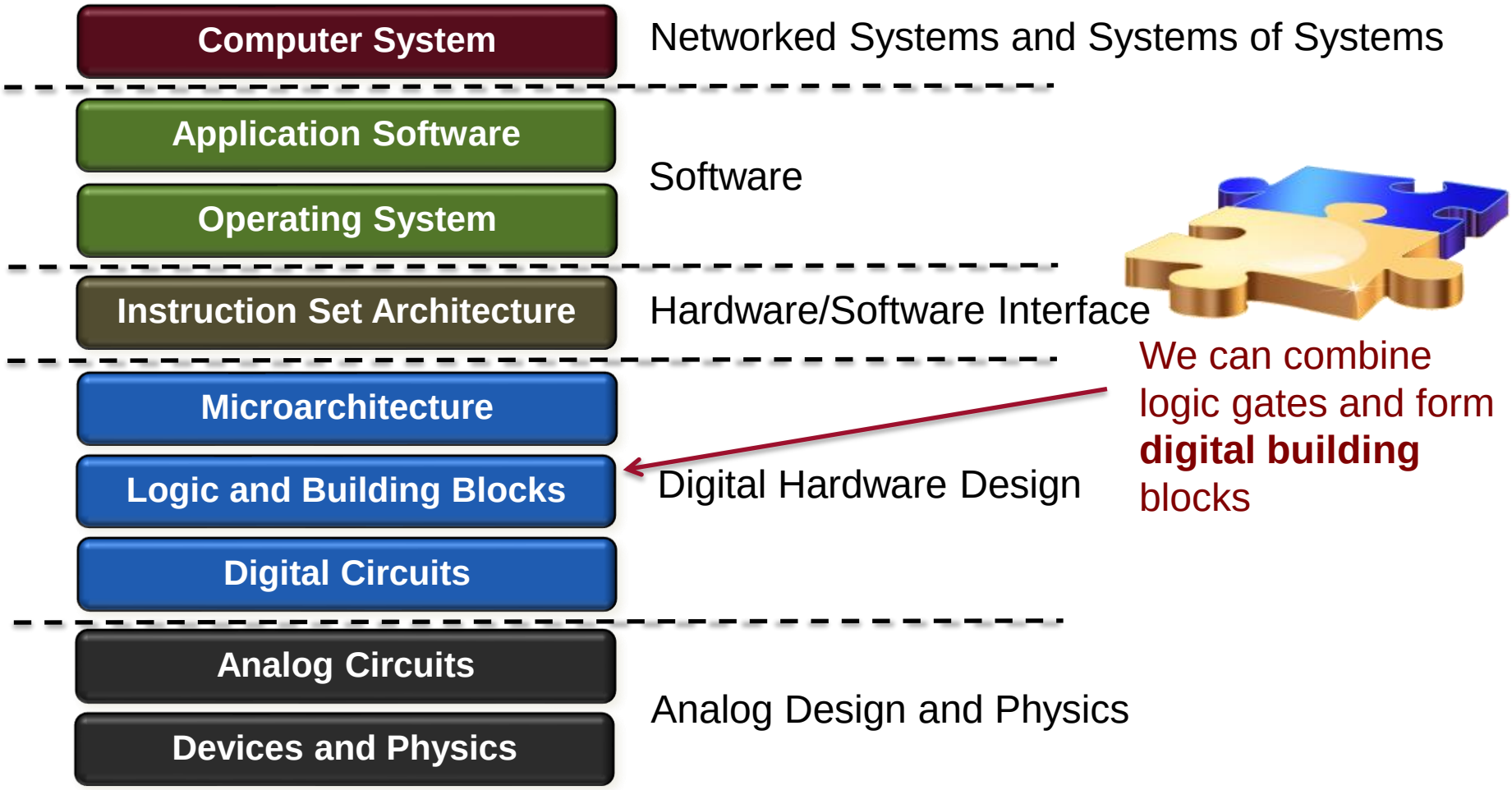
Building Blocks: Multiplexers, Decoders, and Adders



Acknowledgement: The structure and several of the good examples are derived from the book "Digital Design and Computer Architecture" (2013) by D. M. Harris and S. L. Harris.

Artur Podobas
podobas@kth.se

Abstractions in Computer Systems



Combinational Blocks (1/3)
Multiplexers

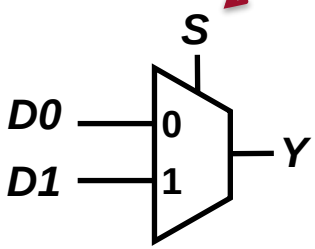
What is this?

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

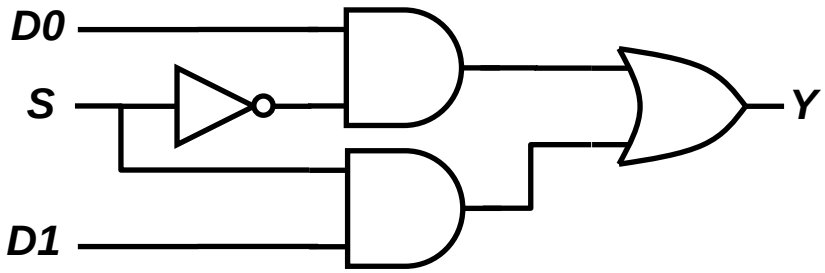
It's a **2:1 Multiplexer**.

2 bits for the data input

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



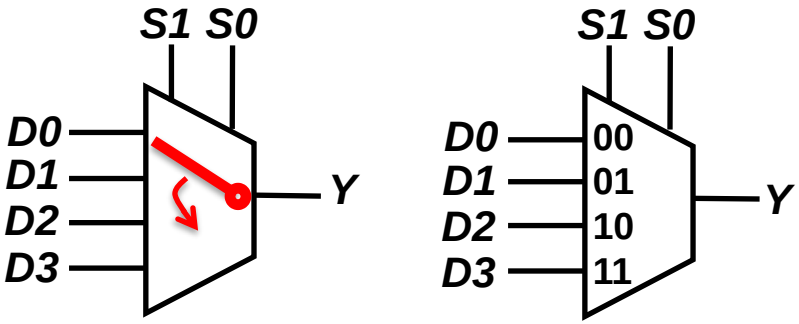
1 output



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.

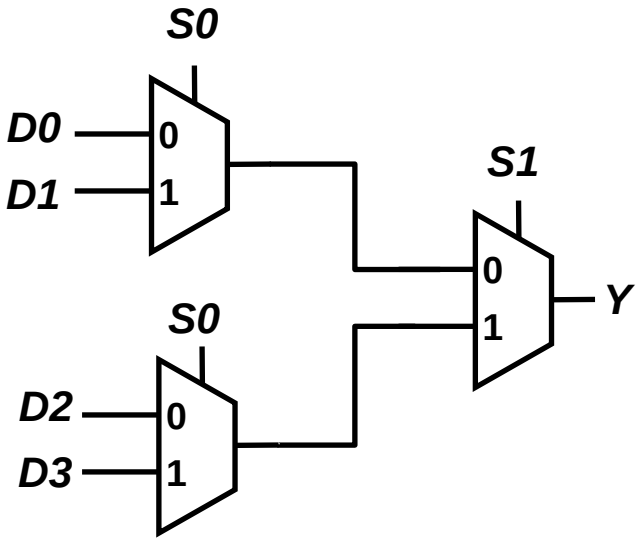


4:1 multiplexer (4 inputs, 1 output).

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

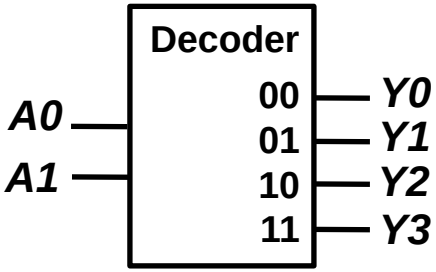
D0 = 1, D1 = 0, D2=1, D3=0,
S1 = 1, S0 = 0 Answer: *Y* = 1

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.



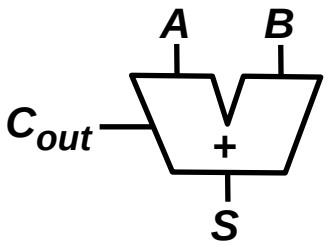
A1	A0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

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

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

Arithmetic Circuits and Numbers (1/7)
Half and Full Adders

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

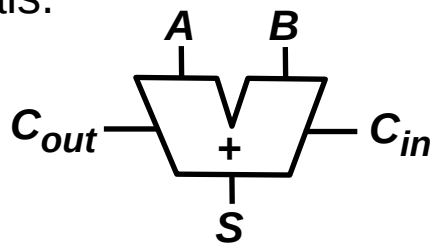


How can we add bigger numbers?

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.



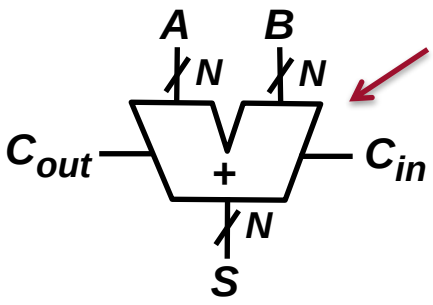
C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Exercise: Complete the truth table

Artur Podobas
podobas@kth.se

Arithmetic Circuits and Numbers (2/7)
Carry Propagate Adders

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



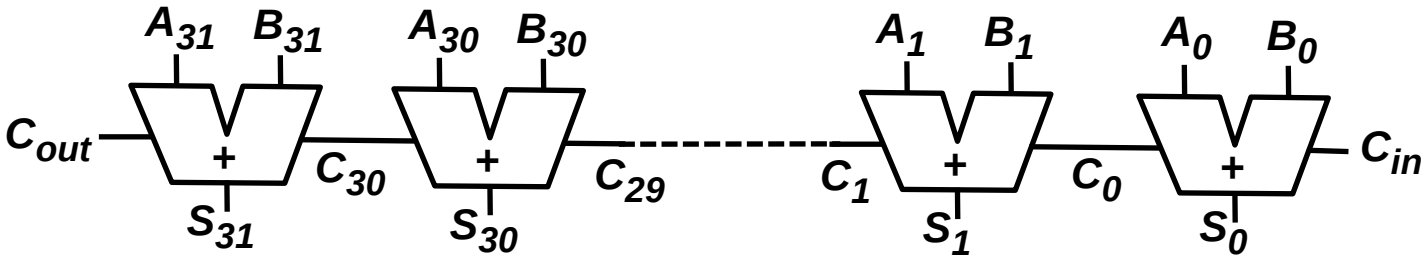
Note the notation for a N-bit bus.

See course book (advanced part)

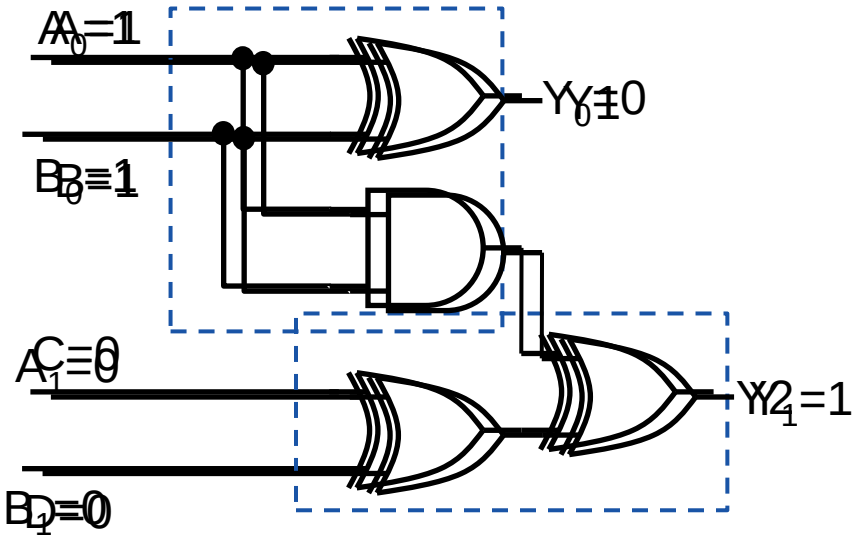
Three common implementations of CPAs are:

- **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



Arithmetic Circuits and Numbers: Remember this circuit?



Written another way:

$A_{1..0}$ = ”01”

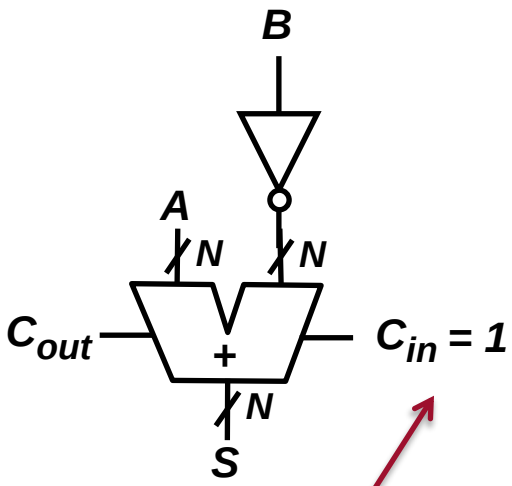
$B_{1..0}$ = + ”01”

$Y_{1..0}$ = ”10”

A 2 bit adder! (without a carry-out for the second bit)

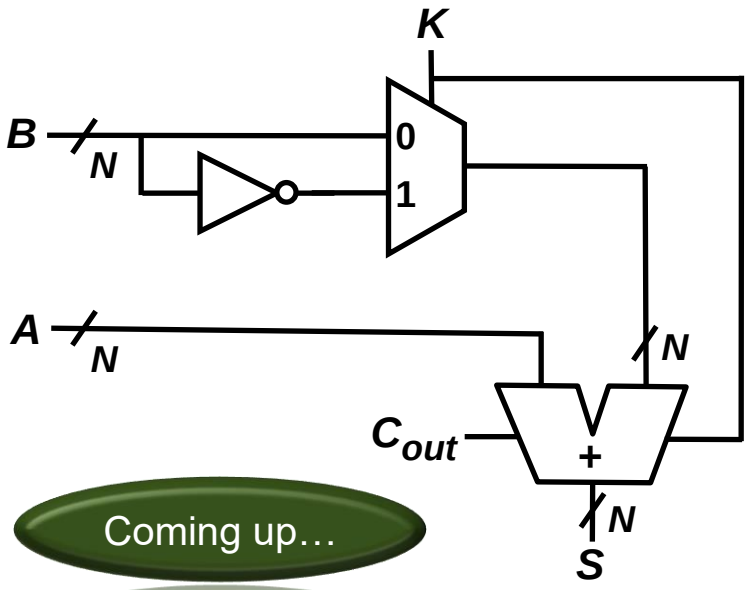
Arithmetic Circuits and Numbers (7/7)
Subtract

Subtract is simple to implemented 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

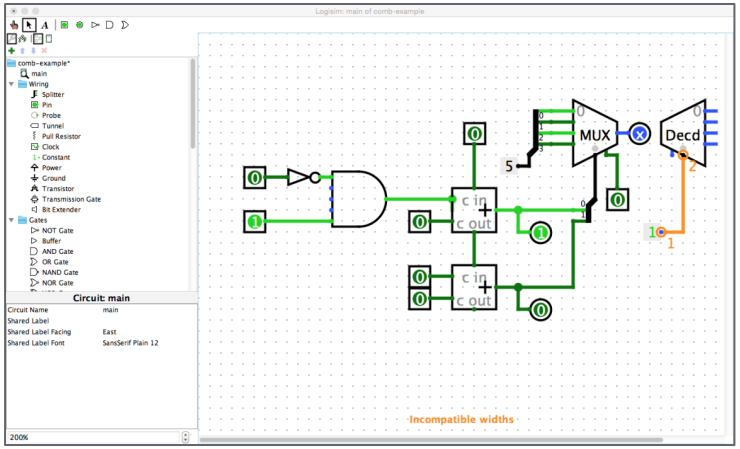


Coming up…

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

Part III

Logisim Demo



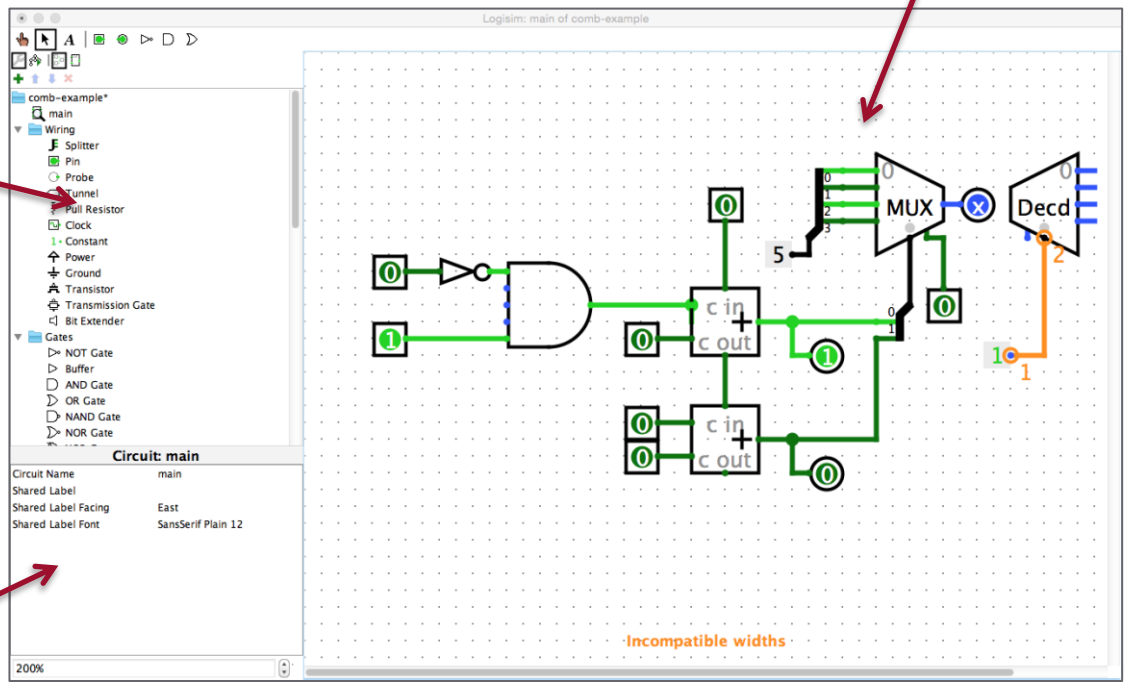
Logisim

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

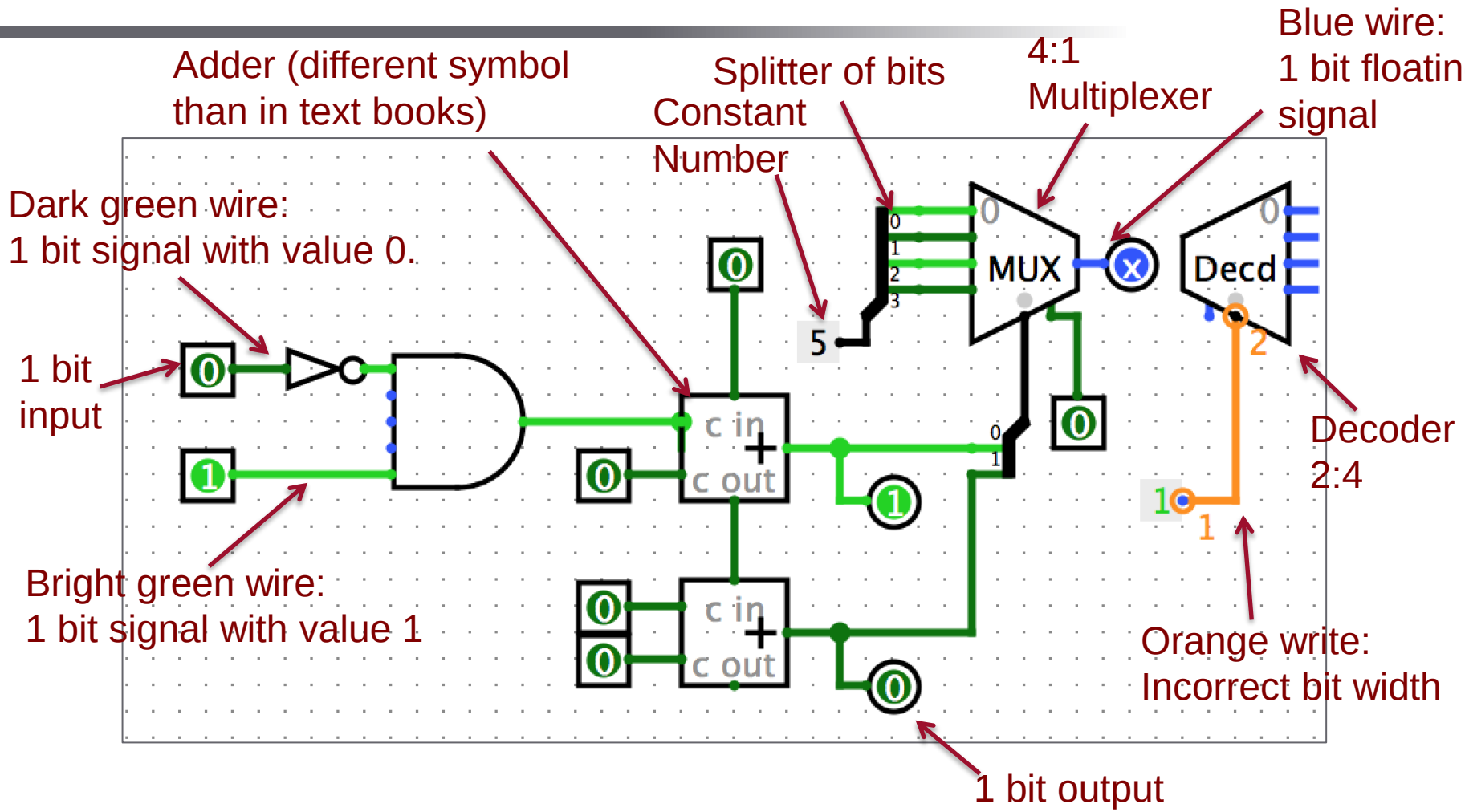
Graphical Model Canvas
Both for construction and simulation

Explorer Pane
Building blocks and gates

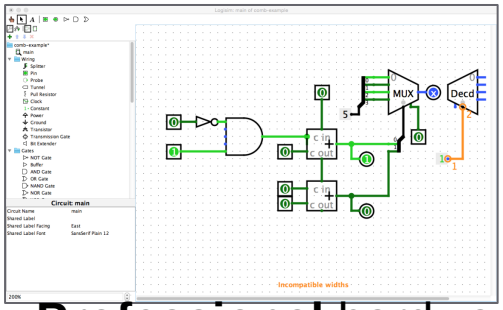
Attribute Table
Configure different components



Some Different Notations in Logisim



Hardware Description Languages



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

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

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

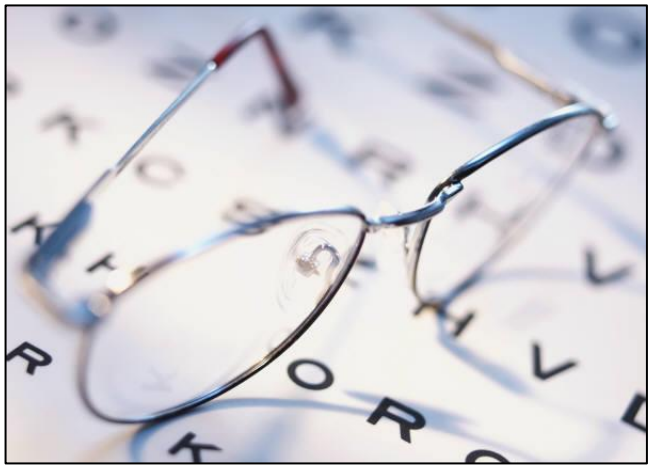
The two most commonly used HDLs in industry are:

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

It also exist recent domain-specific languages (DSLs) for hardware design. Ex. **Chisel** from UC Berkeley (embedded in Scala). Also, there are tools that convert from C-code directly to Hardware; these are called **High-Level Synthesis** tools and are becoming very popular.

We will soon finish!

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.

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!