Digital Logic Design: a rigorous approach ©

Chapter 11: Foundations of combinational circuits

Guy Even   Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

January 14, 2013

Book Homepage:
http://www.eng.tau.ac.il/~guy/Even-Medina

Combinational Circuit

Representation of Boolean functions:

- Truth tables can be implemented by a ROM (e.g., lookup tables, FPGAs)
- SOP Boolean formulas can be implemented by PLA circuits.
- Boolean formulas can be implemented by combinational circuits
- The general case: combinational circuits - topic of this chapter!

Analog approach

- Output is a function of the inputs, provided that the input values do not change for a sufficiently long amount of time.

Our goal now is to define combinational gates. Additional requirements:

- Digital inputs imply digital outputs.
- In the stable state: inputs may fluctuate but must remain digital.

Digital and Logical Signals

1. An analog signal $f : \mathbb{R} \rightarrow \mathbb{R}$.
2. A digital signal $d : \mathbb{R} \rightarrow \{0, 1, \text{non-logical}\}$
3. A digital signal $d(t)$ is logical at time t if $d(t) \in \{0, 1\}$.
4. A digital signal $d(t)$ is stable during interval I if d restricted to I is a constant function whose value is 0 or 1.

Digital view of combinational circuits

Setting and notation:

- To simplify notation, we consider a combinational gate g with 2 inputs, denoted by x_1, x_2, and a single output, denoted by y.
- We refer only to digital signals.
- We denote the digital signal at terminal x_1 by $x_1(t)$. The same notation is used for the other terminals.

Our goals are to:

- specify the functionality of combinational gate g by a Boolean function,
- define when a combinational gate g is consistent, and
- define the propagation delay of g.

We address these goals in reverse order.

Propagation delay

Definition

A combinational gate g is consistent with a Boolean function B at time t if the input values are logical at time t and

$$y(t) = B(x_1(t), x_2(t)).$$

Note that $y(t)$ must be also logical since $x_1(t), x_2(t) \in \{0, 1\}$ and B is a Boolean function.

We attach a Boolean function B to each combinational gate g, namely, B is the functionality of g.

Definition

The propagation delay of a combinational gate g is t_{pd} if the following holds. If the inputs are stable during the interval $[t_1, t_2]$, the gate is consistent with the function B during the interval $[t_1 + t_{pd}, t_2]$.

Propagation delay - remarks

- What if $t_2 < t_1 + t_{pd}$? Periods of steady state must be longer than the propagation delays. Otherwise, the combinational gate may not reach consistency.
- t_{pd} is an upper bound on the amount of time that elapses till a combinational gate becomes consistent (provided that its inputs are stable). The actual time depends on:
  - $x(t)$ during the interval $(-\infty, t)$ (i.e., how fast does the input change?),
  - noise, and
  - manufacturing variance.
- pessimistic assumptions should not render a circuit incorrect (if $t' \geq t_{pd}$, then t' is also a prop. delay).
- Timing analysis of circuits composed of many gates depends on the upper bounds we use; the tighter the bounds, the more accurate the timing analysis is.

Contamination delay

Assume that the combinational gate g is consistent at time t_2, and that at least one input is not stable in the interval (t_2, t_3). We can not assume that the output of g remains stable after t_2. However, in practice, an output may remain stable for a short while after an input becomes instable. We formalize this as follows.

Definition

The *contamination delay* of a combinational device is a lower bound on the amount of time that the output of a consistent gate remains stable after its inputs stop being stable.

We usually make the most "pessimistic" assumption about the contamination delay. Namely, we do not rely on an output remaining stable after an input becomes instable. Formally, we will assume that the contamination delay is zero.

propagation delay and contamination delay

The outputs become stable at most t_{pd} time units after the inputs become stable. The outputs remain stable at least t_{cont} time units after the inputs become instable.

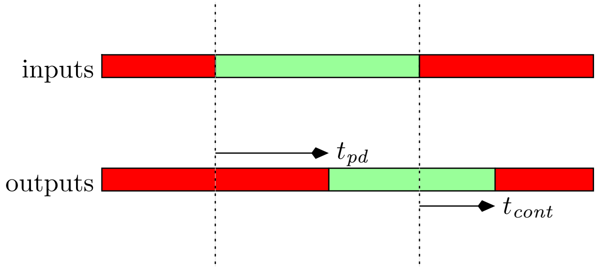


Figure: The x-axis corresponds to time. The red segments signify that the signal is not guaranteed to be logical; the green segments signify that the signal is guaranteed to be stable.

Example

Consider an AND-gate with inputs $x_1(t)$ and $x_2(t)$ and an output $y(t)$. Suppose that the propagation delay of the gate is $t_{pd} = 2$ seconds. (All time units are in seconds in this example, so units will not be mentioned anymore in this example).

- the inputs equal 1 during the interval $[100, 109]$. When is the gate consistent?
- $x_1(t) = 1$ during the interval $(109, 115]$, $x_2(t) =$ non-logical during the interval $(109, 110)$, and $x_2(t) = 0$ during the interval $[110, 115]$. What can we say about $y(t)$?
- $x_2(t)$ remains stable during the interval $[110, 120]$, $x_1(t)$ becomes non-logical during the interval $(115, 116)$, and $x_1(t)$ equals 1 again during the interval $[116, 120]$. What can we say about $y(t)$?

$\text{AND}(0, non - logical) = 0$?

We cannot determine that $\text{AND}(0, non - logical) = 0$. This is technology dependent. Our formalism does not imply this at all! For example, in a CMOS NAND-gate, one can determine that the output is zero if one of the outputs is one (even if the other input is non-logical).
Another drawback of assuming that $\text{AND}(0, non - logical) = 0$ is that timing depends on the values of the signals (timing analysis becomes a very hard computational problem). In particular, instead of a task that can be computed in linear time, timing analysis (of general combinational circuits) becomes an NP-hard task (i.e., a task that is unlikely to be solvable in polynomial time).

Building Blocks

The building blocks of combinational circuits:

- Combinational gates (e.g., inverter, OR-gate, NOR-gate, etc.)
- Wires and nets

combinational gates - terminology

- The basic gates that we consider are: inverter (NOT-gate), OR-gate, NOR-gate, AND-gate, NAND-gate, XOR-gate, NXOR-gate, multiplexer (MUX). All this gates have a single output.
- inputs and outputs of a gate are often referred to as *terminals*, *ports*, or even *pins*.
- fan-in of a gate g = number of input terminals of g (i.e., the number of bits in the domain of the Boolean function that specifies the functionality of g).
- basic gates have constant fan-in (2-3).
- fan-out $\neq$ the number of output ports.

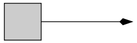
combinational gates - terminology

- $\{in(g)_i\}_{i=1}^{n}$ = the input ports of a gate g, where n =fan-in(g).
- $\{out(g)_i\}_{i=1}^{k}$ = the output ports of a gate g, where k=number of output ports of g.
- $$terminals(g) \triangleq \{in(g)_i\}_{i=1}^{\text{IN}(g)} \cup \{out(g)_i\}_{i=1}^{\text{OUT}(g)}.$$

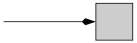
Input/Output gates

Definition (input and output gates)

An *input gate* is a gate with zero inputs and a single output. An *output gate* is a gate with one input and zero outputs.



Input Gate



Output Gate

- Inputs from the "external world" are fed to a circuit via input gates.
- Outputs to the "external world" are fed by the circuit via output gates.
- an input gate is labeled (IN, x_i), where x_i is the name of the signal along the wire that emanates from it.
- an output gate is labeled (OUT, y_i), where y_i is the name of the signal along the wire that enters it.

Wires and nets

A wire is a connection between two terminals (e.g., an output of one gate and an input of another gate). In the zero-noise model, the signals at both ends of a wire are identical.

Very often we need to connect several terminals (i.e., inputs and outputs of gates) together. We could, of course, use any set of edges (i.e., wires) that connects these terminals together. Instead of specifying how the terminals are physically connected together, we use nets.

Definition

A net is a subset of terminals that are connected by wires. The fan-out of a net N is the number of input terminals that are contained in N.

Example

We may draw a net in any way that we find convenient or aesthetic. The interpretation of the drawing is that terminals that are connected by lines or curves constitute a net.

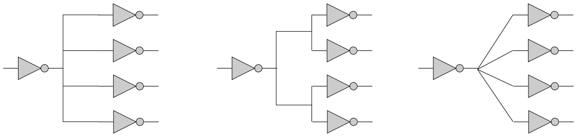


Figure: Three equivalent nets.

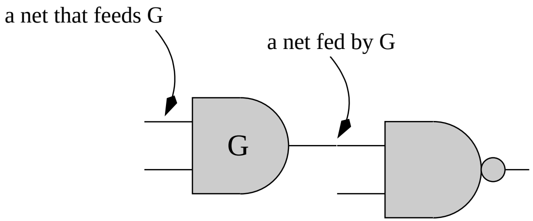
Digital Signal in A Net

How do we define the digital signal $N(t)$ for the whole net?

- Many terminals, perhaps far away, why should they "agree"?
- We solve this problem by defining $N(t)$ as logical only if there is a consensus among all the digital interpretations of the analog signals at all the terminals of the net.
- If there is no consensus, then $N(t)$ is non-logical.

Direction in nets

We say that a net N feeds an input terminal t if the input terminal t is in N.
We say that a net N is fed by an output terminal t if t is in N.
Direction of signals along nets is obtained in "pure" CMOS gates as follows. Output terminals are connected (via low resistance) to the ground or to the power (but not both!). Input terminals, on the other hand, are connected only to capacitors.



Simple nets

The following definition captures the type of nets we would like to use. We call these nets *simple*.

Definition

A net N is *simple* if (i) N is fed by exactly one output terminal, and (ii) N feeds at least one input terminal.

A simple net N that is fed by the output terminal t and feeds the input terminals $\{t_i\}_{i \in I}$ can be modeled by the wires $\{w_i\}_{i \in I}$. Each wire w_i connects t and t_i. In fact, since information flows in one direction, we may regard each wire w_i as a directed edge $t \rightarrow t_i$. To simplify the discussion, we model simple nets by a "star" of wires emanating from a common output terminal.
Each such wire connects an output terminal of a gate to input terminal of a gate. Thus, a full description of a wire is of the form $(g_1, t_1) \longrightarrow (g_2, t_2)$, where t_1 is an output terminal of gate g_1 and t_2 is an input terminal of gate t_2.

Wire notation

Often, we abbreviate and describe the wire $(g_1, t_1) \longrightarrow (g_2, t_2)$ by $g_1 \longrightarrow g_2$. This abbreviation is not ambiguous if the following holds:

- The gate g_1 has a single output terminal. Since there is only one output terminal, we need not specify to which output terminal the wire is connected.
- (i) Only two wires are directed toward g_2, (ii) g_2 has two input terminals, and (iii) the Boolean function of g_2 is commutative. In this case, we connect each wire to a different input terminal.

Library of gates

Let Γ denote a library of combinational gates that contains standard combinational gates such as an inverter, OR-gate, AND-gate, et cetera.
The library Γ contains a sub-library IO that contains two special types of gates: input-gates (IN, x_i) and output-gates (OUT, y_j).

How to define a combinational circuit? (1)

Suppose we want to design a circuit that contains two AND gates, three inputs, x_1, x_2, x_3, and two outputs y_1, y_2, where $y_1 = \text{AND}(x_1, x_2)$ and $y_2 = \text{AND}(x_2, x_3)$. One way to describe the circuit is to draw a schematic. We would like to describe the circuit formally (a schematic is perhaps easy to "read", but hard to argue about).

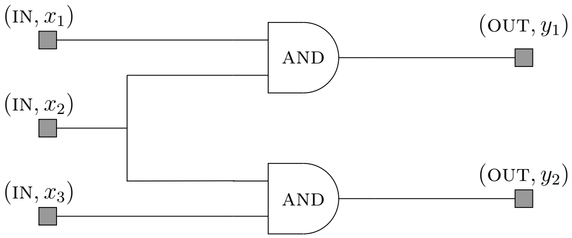
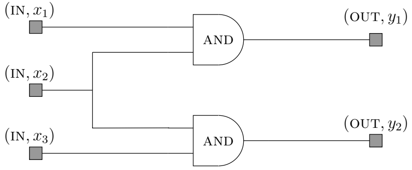


Figure: A combinational circuit.

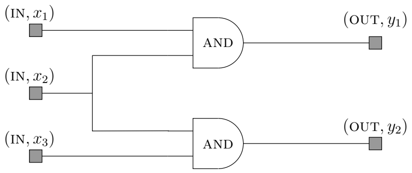
How to define a combinational circuit? (2)



We define a set $V \triangleq \{v_i\}_{i=1}^{7}$ of nodes. Now, we need to assign a gate type to each node. We do this by defining a function $\pi : V \to \Gamma$.

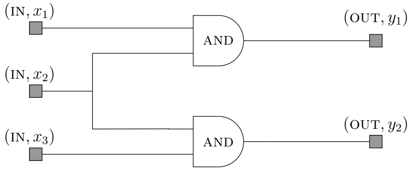
$$\pi(v_1) = (\text{IN}, x_1), \pi(v_2) = (\text{IN}, x_2), \pi(v_3) = (\text{IN}, x_3),$$
$$\pi(v_4) = \pi(v_5) = \text{AND},$$
$$\pi(v_6) = (\text{OUT}, y_1), \pi(v_7) = (\text{OUT}, y_2).$$

How to name terminals?



- Both v_4 and v_5 are assigned AND-gates.
- How can we distinguish between the input ports of v_4 and the input ports of v_5?
- An AND-gate has two input ports, called $in(\text{AND})_1$ and $in(\text{AND})_2$, and one output terminal called $out(\text{AND})$.
- Use "family" names to terminals. For example, the first input port of v_4 is called $(v_4, in(\text{AND})_1)$. This is a bit cumbersome but unambiguous.

How to name terminals?



- In the case of input and output gates, we abbreviate.
- Write (IN, x_i), instead of, $out((\text{IN}, x_i))$.
- Similarly, we write (OUT, y_j), instead of, $in((\text{OUT}, y_j))$.

Definition of Terminals

Consider a set of nodes V and a function $\pi : V \rightarrow \Gamma$ assigns a gate type to each node.

Definition

The set of terminals of V with respect to π is defined as follows

$$\mathit{terminals}(V, \pi) \triangleq \{(v, t) : v \in V, t \in \mathit{terminals}(\pi(v))\}.$$

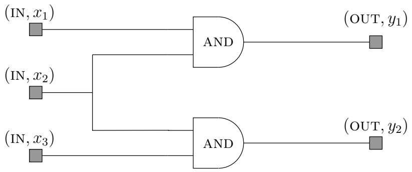
Netlist

A netlist is a way to describe how gates are connected to each other.

Definition

A netlist is a tuple $H = (V, N, \pi)$, where V is a set of nodes, $\pi : V \rightarrow \Gamma$ assigns a gate type to each node, and N is a set of nets over $\mathit{terminals}(V, \pi)$. We require that the nets in N are pairwise disjoint.

Netlist



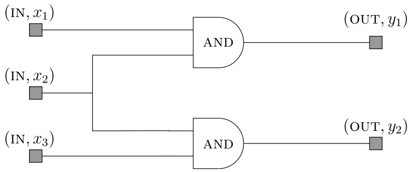
The set N of nets consists of the following nets.

$$\begin{aligned}
&\{(v_1, (\mathrm{IN}, x_1)), (v_4, in(\mathrm{AND})_1)\}\ ,\\
&\{(v_2, (\mathrm{IN}, x_2)), (v_4, in(\mathrm{AND})_2), (v_5, in(\mathrm{AND})_1)\}\ ,\\
&\{(v_3, (\mathrm{IN}, x_3)), (v_5, in(\mathrm{AND})_2)\}\ ,\\
&\{(v_4, out(\mathrm{AND})), (v_6, (\mathrm{OUT}, y_1))\}\ ,\\
&\{(v_5, out(\mathrm{AND})), (v_7, (\mathrm{OUT}, y_2))\}\ .
\end{aligned}$$

Graph Representation of a Netlist with Simple Nets

- A netlist $H = (V, N, \pi)$ in which all nets are simple can be represented by a directed graph $DG(H) = (V, \tilde{N})$.
- For each net $n = \{t, t_1, \ldots, t_k\}$ with an output terminal t and input terminals $t_1, \ldots, t_k$. Suppose that t is a terminal of node v, and t_i is a terminal of node v_i.
- This net n is represented in $\tilde{N}$ by the set of directed edges $\{(v, v_i)\}_{i=1}^{k}$.

Graph Representation of a Netlist with Simple Nets



Set $V = \{v_1, \ldots, v_7\}$ and

$$\tilde{N} = \{(v_1, v_4), (v_2, v_4), (v_2, v_5), (v_3, v_5), (v_4, v_6), (v_5, v_7)\} .$$

Graph representation of a netlist

- $DG(H)$ may have directed edges of the form (v,v); such edges are called self-loops. Self-loops can be obtained by gates that their output is connected to their input.
- $DG(H)$ may have parallel edges.

Definition of Combinational Circuits.

Definition

A netlist $H = (V, N, \pi)$ is a combinational circuit if it satisfies the following conditions.

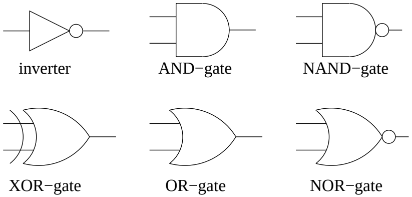
1. Every net in N is simple.
2. Every terminal in $terminals(V, \pi)$ belongs to exactly one net in N.
3. The directed graph $DG(H)$ is acyclic.

Question

Can you check if a netlist is a combinational circuit?

Gate Symbols

Instead of writing the label $\pi(v)$ in the vertex v, one sometimes depicts the vertex by a symbol that represents $\pi(v)$.



Example : Half Adder

The combinational circuit $C = (G, \pi)$ is called a Half-Adder.

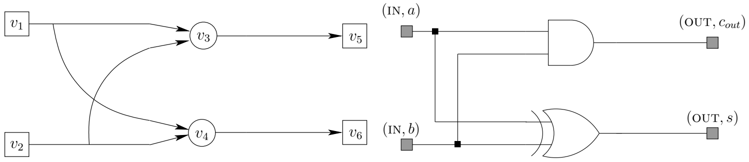


Figure: A Half-Adder combinational circuit and its matching DAG.

The set of the combinational gates in this example is $\Gamma = \{\text{AND}, \text{XOR}\}$. The labeling function $\pi : V \rightarrow \Gamma \cup IO$ is as follows.

$$\begin{aligned} \pi(1) &= (\text{IN}, a), & \pi(2) &= (\text{IN}, b), \\ \pi(3) &= \text{AND}, & \pi(4) &= \text{XOR}, \\ \pi(5) &= (\text{OUT}, c_{out}), & \pi(6) &= (\text{OUT}, s) . \end{aligned}$$

Bad Circuits

Can you explain why these are not valid combinational circuits?

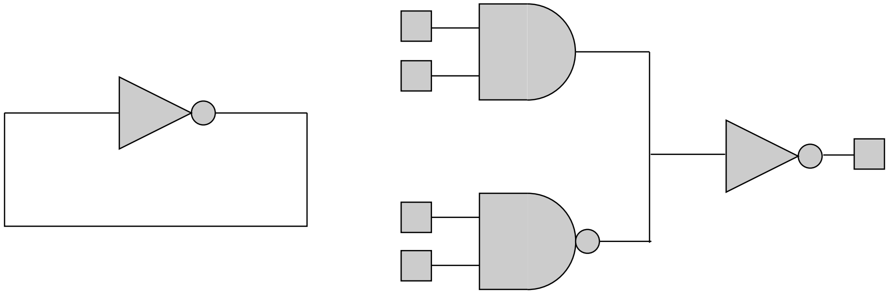


Figure: Two examples of non-combinational circuits.

Important properties of combinational circuits

Completeness: For every Boolean function B, there exists a combinational circuit that implements B.

Soundness: Every combinational circuit implements a Boolean function.

Simulation: Given the digital values of the inputs of a combinational circuit, one can simulate the circuit efficiently (the running time is linear in the size of the circuit). Namely, one can compute the digital values of the outputs of the circuit that are output by the circuit once the circuit becomes consistent.

Delay analysis: Given the propagation delays of all the gates in a combinational circuit, one can compute in linear time an upper bound on the propagation delay of the circuit.

Plan

- algorithm for simulation and delay analysis.
- algorithm implies soundness.
- prove completeness by implementing Boolean formulas.

Assumptions

To simplify matters, assume that every combinational gate:

- has a single output terminal
- has at most two input terminals (fan-in ≤ 2)
- implements a commutative Boolean function.

Reason: port information of each wire can be easily deduced...

Notation

Consider a combinational circuit $C = (G, \pi)$.

- We identify a vertex v with its output terminal, and denote the digital signal at the output terminal of v simply by $v(t)$.
- For an output-gate v, we denote the digital signal at the input terminal of v also by $v(t)$.
- We assume that C has k input gates named them $x_i, \ldots, x_k$.
- To simplify notation, we use $\vec{x}(t)$ to denote the vector $x_1(t), \ldots, x_k(t)$.

Simulation theorem of combinational circuits

Theorem

Assume that the digital signals $\{x_i(t)\}_{i=1}^{k}$ *are stable during the interval* $[t_1, t_2]$. *Then, for every vertex* $v \in V$ *there exist:*

1. *a Boolean function* $f_v : \{0,1\}^k \rightarrow \{0,1\}$, *and*
2. *a propagation delay* $t_{pd}(v)$

such that $v(t) = f_v(\vec{x}(t))$, *for every* $t \in [t_1 + t_{pd}(v), t_2]$.

Note that $t_{pd}(v) \neq t_{pd}(\pi(v))$. The propagation delay $t_{pd}(\pi(v))$ refers to the delay of a single gate of type $\pi(v)$. This delay is measured with respect to the input of the gate. On the other hand, the propagation delay $t_{pd}(v)$ refers to the delay of the output of v with respect to the input gates of the circuit C.

Proof by Algorithm

Simulation algorithm:

- Similar to EVAL algorithm.
- Sorts vertices in topological order.
- Given $\vec{x}$ evaluates value of every output terminal (and therefore, wire).
- Computes accumulated delay along longest paths.

Algorithm 1 SIM$(C, \vec{x})$ - An algorithm for simulating the combinational circuit $C = (G, \pi)$ with respect an input vector $\vec{x}$.

```
• $(v_1, v_2, \ldots, v_n) \leftarrow TS(G)$ {topological sorting of $G$}
• For $i = 1$ to $n$ do
    switch $deg_{in}(v_i)$
        case $deg_{in}(v_i) = 0$:    $\{\pi(v_i) = (\text{IN}, x_j)\}$
            • Let $x_j$ denote the name of $v_i$ before topological
              sorting.
            • Set $f_{v_i}(\vec{x}) \stackrel{\triangle}{=} x_j$ and $t_{pd}(v_i) \stackrel{\triangle}{=} 0$.
```

Algorithm 2 SIM($C, \vec{x}$) - An algorithm for simulating the combinational circuit $C = (G, \pi)$ with respect an input vector $\vec{x}$.

```
• $(v_1, v_2, \ldots, v_n) \leftarrow TS(G)$ {topological sorting of $G$}
• For $i = 1$ to $n$ do
    switch $deg_{in}(v_i)$
        case $deg_{in}(v_i) = 1$:
        If $\{\pi(v_i) = \text{NOT}\}$, then
            • Let $v_j \longrightarrow v_i$ denote the arc that enters $v_i$.
            • Set $f_{v_i}(\vec{x}) = \text{NOT}(f_{v_j}(\vec{x}))$ and
              $t_{pd}(v_i) = t_{pd}(v_j) + t_{pd}(\text{NOT})$.
        If $\{\pi(v_i) = (\text{OUT}, y\}$, then
            • Let $v_j \longrightarrow v_i$ denote the arc that enters $v_i$.
            • Set $f_{v_i}(\vec{x}) = f_{v_j}(\vec{x})$ and $t_{pd}(v_i) = t_{pd}(v_j)$.
```

Algorithm 3 SIM$(C, \vec{x})$ - An algorithm for simulating the combinational circuit $C = (G, \pi)$ with respect an input vector $\vec{x}$.

```
• $(v_1, v_2, \ldots, v_n) \leftarrow TS(G)$ {topological sorting of $G$}
• For $i = 1$ to $n$ do
    switch $deg_{in}(v_i)$
          case $deg_{in}(v_i) = 2$:
              • Let $v_j \longrightarrow v_i$ and $v_k \longrightarrow v_i$ denote the arcs that
                enter $v_i$.
              • Set $f_{v_i}(\vec{x}) = B_{\pi(v_i)}(f_{v_j}(\vec{x}), f_{v_k}(\vec{x}))$, and
                $t_{pd}(v_i) = \max\{t_{pd}(v_j), t_{pd}(v_k)\} + t_{pd}(\pi(v_i))$.
```

Algorithm Correctness

$$\forall i \in [1..n] \; \forall \vec{x} \in \{0,1\}^k \; \forall t \in [t_1 + t_{pd}(v_i), t_2] \quad : \quad v_i(t) = f_{v_i}(\vec{x}).$$

The proof is by induction on i, the index of a vertex after topological sorting takes place. We assume that topological ordering orders the sources first, namely, $v_i = x_i$, for $1 \leq i \leq k$.

Multiple topological orderings?!

Recall that a DAG may have more than one topological ordering.

Lemma

The output of $SIM(C, \vec{x})$ does not depend on the topological ordering computed by $TS(G)$.

Soundness

- The simulation Theorem enables us to regard a combinational circuit as a “macro-gate”.
- This macro-gate computes a Boolean function $B : \{0,1\}^k \to \{0,1\}^\ell$, where k denotes the number of input gates and ℓ denotes the number of output gates.
- All instances of the same combinational circuit implement the same Boolean function and have the same propagation delay.

Corollary (Soundness)

Every combinational circuit implements a Boolean function.

Remarks

- Simulation algorithm generalizes EVAL from trees to DAGs.
- The computation of the propagation delays is, in fact, a computation of longest paths in DAGs with non-unit delays $\delta : V \longrightarrow \mathbb{R}^{\geq 0}$.

Algorithm 4 weighted-longest-path-lengths(V, E, δ) - An algorithm for computing the longest delays of paths in a DAG. Returns a delay function $d(v)$.

```
1 topological sort: $(v_0, \ldots, v_{n-1}) \leftarrow TS(V, E)$.
2 For $j = 0$ to $(n-1)$ do
    1 If $v_j$ is a source then $d(v_j) \leftarrow \delta(v_j)$.
    2 Else
          $d(v_j) = \delta(v_j) + \max\left\{d(v_i) \mid i < j \text{ and } (v_i, v_j) \in E\right\}.$
```

What about constant inputs?

- We do not rule out the usage of constants as inputs.
- In this case we add the possibility for input-gates labeled $(\text{IN}, 0)$ and $(\text{IN}, 1)$. Such an input gate feeds a constant to the circuit.
- Algorithm SIM needs to be modified to handle constant inputs. Namely, the case that v_i is a source has to be split to a constant input and a variable input.

Completeness

- Want to prove that every Boolean function can be implemented by a combinational circuit.
- $\{\neg, \text{OR}, \text{AND}\}$ is a complete set of logical connectives.
- Given a Boolean function $B : \{0,1\}^n \rightarrow \{0,1\}$, represent it by a Boolean formula φ.
- We need to show how to implement φ by a combinational circuit.

From formulas to circuits

Demonstrate by example (full proof in lecture notes).

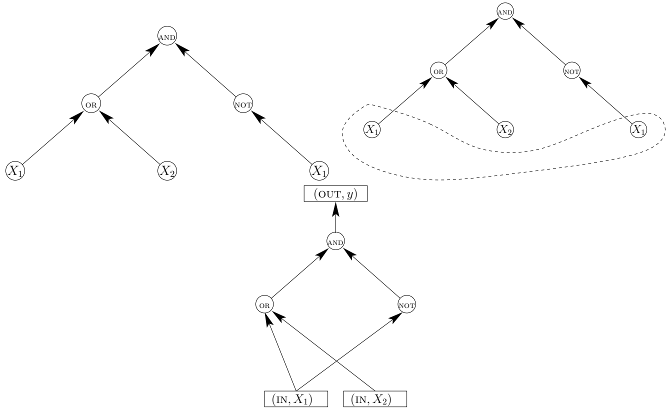


Figure: (a) the parse tree of φ, (G, π), (b) merge sources labeled by same variable, (c) combinational circuit that implements φ.

From formulas to circuits

- Take parse tree of φ.
- Merge sources labeled by the same variable.
- Theorem: tree becomes a DAG.
- label sources by input gates.
- Add root & label it by output gate.
- Voilà!

Cost

- Let $C = (G, \pi)$ denote a combinational circuit where $G = (V, E)$ is a directed graph and $\pi : V \to \Gamma \cup IO$ is a labeling.
- Let $c : \Gamma \cup IO \to \mathbb{R}^{\geq 0}$ denote a cost function. Usually, input-gates and output-gates have zero cost.

Definition

The cost of C is defined by

$$c(C) \triangleq \sum_{v \in V} c(\pi(v)).$$

Propagation delay

The propagation delays $t_{pd}(v)$ are computed by Algorithm SIM$(C, \vec{x})$.

Definition

The propagation delay of C is defined by

$$t_{pd}(C) \triangleq \max_{v \in V} t_{pd}(v).$$

We often refer to the propagation delay of a combinational circuit as its depth or simply its delay.

Definition

The propagation delay of a path p in G is defined as

$$t_{pd}(p) \triangleq \sum_{v \in p} t_{pd}(\pi(v)).$$

Critical paths

Algorithm $\mathrm{SIM}(C, \vec{x})$ computes the largest delay of a path in G.

Claim

$$t_{pd}(C) = \max\{t_{pd}(p) \mid p \text{ is a path in } G\}$$

Definition

Let $G = (G, \pi)$ denote a combinational circuit. A path p in G is critical if $t_{pd}(p) = t_{pd}(C)$.

We focus on critical paths that are maximal (i.e., cannot be further augmented). This means that maximal critical paths begin in an input-gate and end in an output-gate.

relative gate costs and delay

Müller and Paul compiled a table of costs and delays of gates. Obtained by considering ASIC libraries of two technologies and normalizing them with respect to the cost and delay of an inverter.

Gate	Motorola		Siemens-Venus	
	cost	delay	cost	delay
INV	1	1	1	1
AND,OR	2	2	2	1
NAND, NOR	2	1	2	1
XOR, NXOR	4	2	6	2
MUX	3	2	3	2

Table: Relative costs and delays of gates (normalized to inverters)

Semantics and Syntax

- semantics - the function that a circuit implements (functionality, behavior). In non-combinational circuits, the output depends not only on the current inputs, so semantics cannot be described simply by a Boolean function.
- syntax - a formal set of rules that govern how "grammatically correct" circuits are constructed from smaller circuits (just as sentences are built by combining words).
  - the functionality (or gate-type) of each gate is not important.
  - rules for connecting gates together must be followed.
  - syntax does not guarantee that the resulting circuit is useful.
  - syntax is a restriction that brings many benefits: well defined functionality, simple simulation, and simple timing analysis.

Design Rules

In this chapter we defined design rules for building combinational circuits. These design rules define syntactically correct circuits. Our main result is that syntactically correct circuits, called combinational circuits, can implement any Boolean function.

We are now left with the following design task: Given a Boolean function B, design a combinational circuit C that implements B such that the delay and cost of C is as small as possible.

Summary

- Combinational circuits: formal definition.
- Bottom-up approach: basic building blocks: gates and wires. Each gate has a simple specification: functionality and t_{pd}.
- Syntactic definition of combinational circuits: only depends on the topology of the circuit, namely, how the terminals of the gates are connected.
  One can check in linear time whether a given circuit is indeed a combinational circuit.
- Easy simulation: one can compute in linear time the digital signals of every wire in the circuit. Moreover, one can also compute in linear time the propagation delay of every wire.
- Two quality measures: cost and propagation delay. The cost of a combinational circuit is the sum of the costs of the gates in the circuit. The propagation delay of a combinational is the maximum delay of a path in the circuit.