Intro. to Computer Architecture

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨csdsp@bristol.ac.uk⟩

January 9, 2018

Keep in mind there are *two* PDFs available (of which this is the latter):

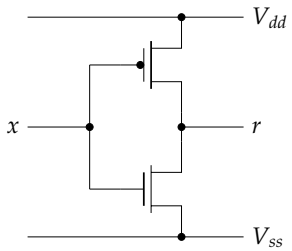
1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

- Question: what does this organisation of MOSFET transistors *do*?

Circuit

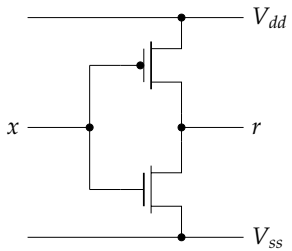


Notes:

COMS12200 lecture: week #3

- Question: what does this organisation of MOSFET transistors *do*?

Circuit



Answer:

- Connect x to V_{ss}:
  1. the top P-MOSFET will be connected,
  2. the bottom N-MOSFET will be disconnected,
  3. r will be connected to V_{dd}.
- Connect x to V_{dd}:
  1. the top P-MOSFET will be disconnected,
  2. the bottom N-MOSFET will be connected,
  3. r will be connected to V_{ss}.

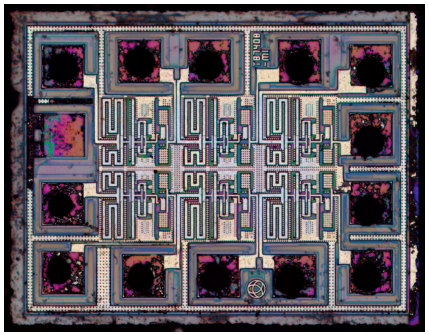
i.e., this matches the behaviour of NOT: it's an **inverter**.

Notes:

© Daniel Page (<redacted>)
Intro. to Computer Architecture
git # 3b6f641 @ 2018-01-09
University of BRISTOL

- Question: what does this organisation of MOSFET transistors *do*?

Circuit



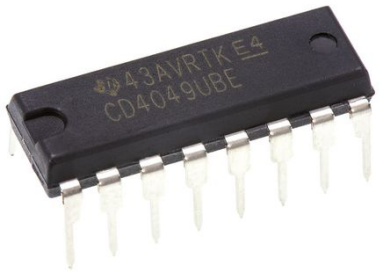
Notes:

http://zeptobars.com/en/read/CD4049-cmos-inverter-metal-gate

COMS12200 lecture: week #3

- Question: what does this organisation of MOSFET transistors *do*?

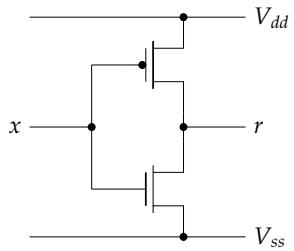
Circuit



Notes:

- Question: what does this organisation of MOSFET transistors *do*?

Circuit



- Bad news: designing complex functionality using transistors alone is difficult, since transistors are *too* low-level.
- Good news: we can organise various types of simple functionality into (reusable) high*er*-level **logic gates**.

Notes:

Logic Gates (1)

Hang on! This sounds *very* familiar if we

1. form a
   - **pull-up network** of P-MOSFET transistors connected to V_{dd},
   - **pull-down network** of N-MOSFET transistors connected to V_{ss},
2. assume power rails are everywhere, and relabel

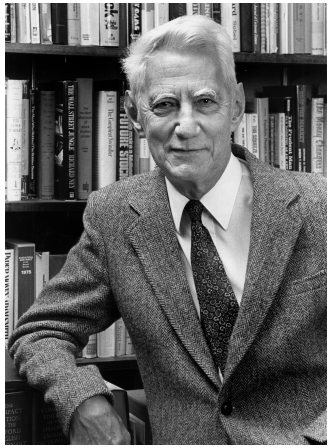
$$\begin{array}{lcll} V_{ss} & = & 0\text{V} \simeq GND & \models 0 \\ V_{dd} & = & 5\text{V} & \models 1 \end{array}$$

   and
3. describe the operation of each logic gate using a truth table, e.g.,

NOT	
x	r
0	1
1	0

NAND		
x	y	r
0	0	1
0	1	1
1	0	1
1	1	0

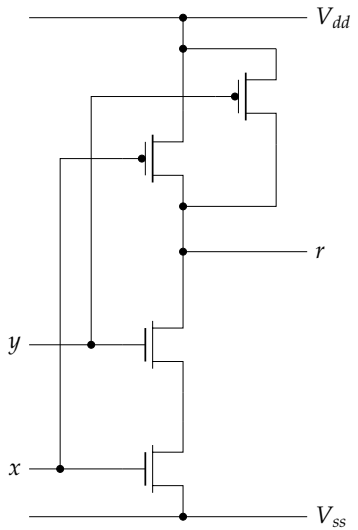
NOR		
x	y	r
0	0	1
0	1	0
1	0	0
1	1	0



Notes:

Logic Gates (2)
A NAND gate

Circuit

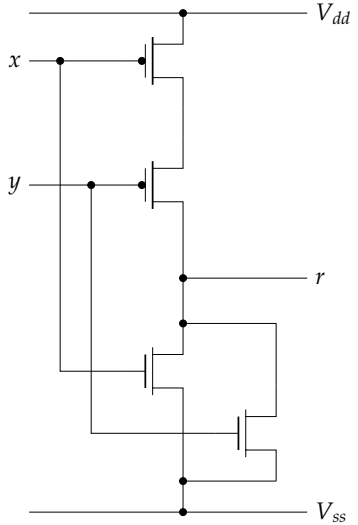


- Connect both x and y to V_{ss}:
  1. both top P-MOSFETs will be connected,
  2. both bottom N-MOSFETS will be disconnected,
  3. r will be connected to V_{dd}.
- Connect x to V_{dd} and y to V_{ss}:
  1. the right-most P-MOSFET will be connected,
  2. the upper-most N-MOSFET will be disconnected,
  3. r will be connected to V_{dd}.
- Connect x to V_{ss} and y to V_{dd}:
  1. the left-most P-MOSFET will be connected,
  2. the lower-most N-MOSFET will be disconnected,
  3. r will be connected to V_{dd}.
- Connect both x and y to V_{dd}:
  1. both top P-MOSFETs will be disconnected,
  2. both bottom N-MOSFETS will be connected,
  3. r will be connected to V_{ss}.

Notes:

Logic Gates (3)
A NOR gate

Circuit

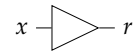
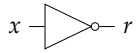
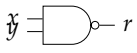
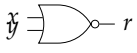
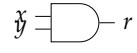
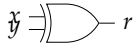


- Connect both x and y to V_{ss}:
  1. both top P-MOSFETs will be connected,
  2. both bottom N-MOSFETS will be disconnected,
  3. r will be connected to V_{dd}.
- Connect x to V_{dd} and y to V_{ss}:
  1. the upper-most P-MOSFET will be disconnected,
  2. the left-most N-MOSFET will be connected,
  3. r will be connected to V_{ss}.
- Connect x to V_{ss} and y to V_{dd}:
  1. the lower-most P-MOSFET will be disconnected,
  2. the right-most N-MOSFET will be connected,
  3. r will be connected to V_{ss}.
- Connect both x and y to V_{dd}:
  1. both top P-MOSFETs will be disconnected,
  2. both bottom N-MOSFETS will be connected,
  3. r will be connected to V_{ss}.

Notes:

Logic Gates (4)

Definition

r is x	$\equiv$	$r = x$	$\equiv$	
r is NOT x	$\equiv$	$r = \neg x$	$\equiv$	
r is x NAND y	$\equiv$	$r = x \overline{\wedge} y$	$\equiv$	
r is x NOR y	$\equiv$	$r = x \overline{\vee} y$	$\equiv$	
r is x AND y	$\equiv$	$r = x \wedge y$	$\equiv$	
r is x OR y	$\equiv$	$r = x \vee y$	$\equiv$	
r is x XOR y	$\equiv$	$r = x \oplus y$	$\equiv$	

Notes:

Physical Limitations (1)

Delay

Notes:

Definition

Within some combinatorial logic, two classes of **delay** (which is often described as **propagation delay**, with a hint toward delay of signals more generally) dictate the time between change to some input and corresponding change (if any) in an output: these are

- **wire delay**, which relates to the time taken for current to move through the conductive wire from one point to another, and
- **gate delay**, which relates to the time taken for transistors in each gate to switch between connected and unconnected states.

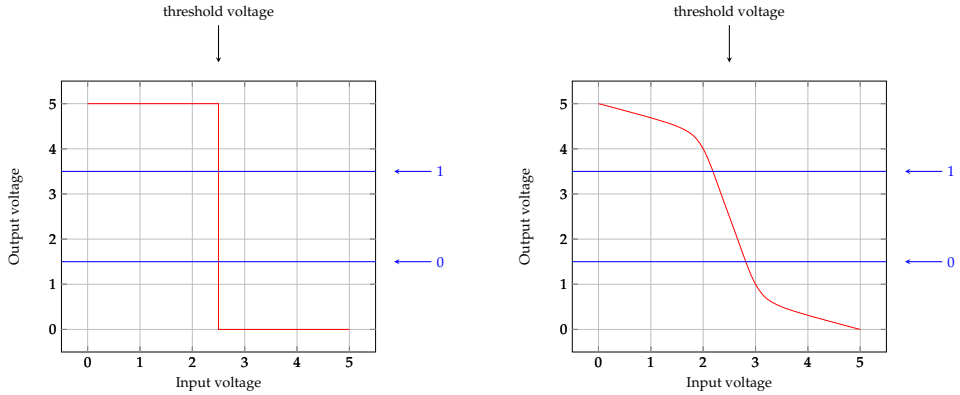
The latter is typically larger than the former, and both relate to the associated implementations: the latter relates to properties of the transistors used, the former to properties of the wire (e.g., conductivity, length, and so on).

Definition

The **critical path** through some combinatorial logic is the longest sequential sequence of delays (so wire and/or gate delays) between the inputs and outputs.

Physical Limitations (2)
Delay

- Consider behaviour of the MOSFET-based NOT gate



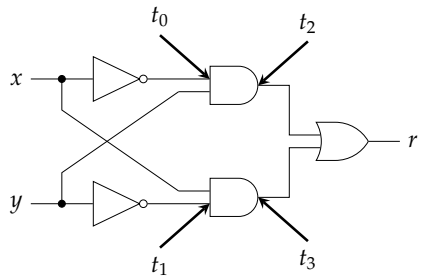
st.

- the left-hand side illustrates an idealised, square response, whereas
- the left-hand side illustrates a (more) realistic, curved response.

Notes:

Physical Limitations (3)
Delay

Circuit



Example

Consider setting $x = 0$ and $y = 1$; the circuit computes

$$\begin{array}{lcll} x & = & & = 0 \\ y & = & & = 1 \\ t_0 & = & \neg x & = 1 \\ t_1 & = & \neg y & = 0 \\ t_2 & = & t_0 \wedge y & = 1 \\ t_3 & = & t_1 \wedge x & = 0 \\ r & = & t_2 \vee t_3 & = 1 \end{array}$$

but this is a *static* view of computation in the sense that we assume all signals eventually just propagate through the circuit.

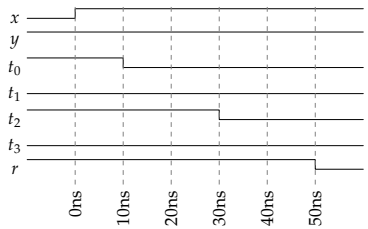
Notes:

Example

Including gate delay gives a *dynamic* view computation; imagine the delay of

1. a NOT gate is 10ns,
2. an AND gate is 20ns, and
3. an OR gate is 20ns

and then flip $x = 0, y = 1$ to $x = 1, y = 1$:

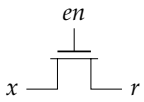


Note that

1. the circuit takes some time matching the **critical path** (i.e, 50ns through a NOT, an AND and an OR gate) to settle into the right state, and as a result
2. at some points in time, the output doesn't match the inputs.

Notes:

- It is attractive to use 3**-state logic**, introducing an "extra" pseudo-value, i.e.,
  1. 0 represents **false**,
  2. 1 represents **true**, and
  3. Z represents **high impedance**.
- Think of high impedance as being the null value; the idea is to allow a wire to be "disconnected" per



Enable		
x	en	r
0	0	**Z**
1	0	**Z**
Z	0	**Z**
0	1	0
1	1	1
Z	1	**Z**
0	**Z**	**Z**
1	**Z**	**Z**
Z	**Z**	**Z**

st. the Enable gate is really just a switch.

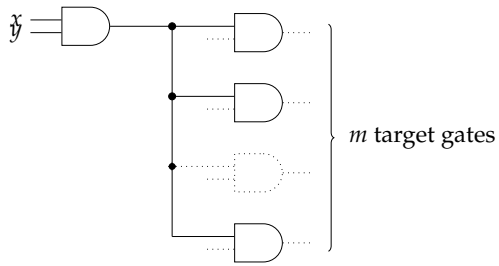
Notes:

Definition

Consider a given logic gate:

- The term **fan-in** is used to describe the number of inputs to a given gate.
- The term **fan-out** is used to describe the number of inputs (so in a rough sense the number of *other* gates) the output of a given gate is connected to.

Example



Notes:

Conclusions

- Take away points:
  1. We've now got logic gates where it's clear
     - what their *behavioural* properties are since they relate directly to transistors, *and*
     - what their *functional* properties are since they relate directly to Boolean algebra,

     i.e., there is no "magic" going on behind the scenes.
  2. The tough bit is the design and optimisation challenge; that is, how do we now design something to
     - match some specification, e.g., "implement some function f", *and*
     - match some design goals, e.g., "use less than X gates (or Y transistors)" or "ensure a delay less than Zns".

Notes:

Additional Reading

- *Wikipedia: Logic gate*. URL: http://en.wikipedia.org/wiki/Logic_gate.
- D. Page. "Chapter 2: Basics of digital logic". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer-Verlag, 2009.
- R.J. Smith and R.C. Dorf. "Chapter 13: Logic elements". In: *Circuits, Devices and Systems*. 5th ed. John Wiley, 1992.
- W. Stallings. "Chapter 11: Digital logic". In: *Computer Organisation and Architecture*. 9th ed. Prentice-Hall, 2013.
- A.S. Tanenbaum and T. Austin. "Section 3.1: Gates and Boolean algebra". In: *Structured Computer Organisation*. 6th ed. Prentice-Hall, 2012.
- A.S. Tanenbaum and T. Austin. "Section 3.2.1: Integrated circuits". In: *Structured Computer Organisation*. 6th ed. Prentice-Hall, 2012.

Notes:

References

[1] *Wikipedia: Logic gate*. URL: http://en.wikipedia.org/wiki/Logic_gate (see p. 37).

[2] D. Page. "Chapter 2: Basics of digital logic". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer-Verlag, 2009 (see p. 37).

[3] R.J. Smith and R.C. Dorf. "Chapter 13: Logic elements". In: *Circuits, Devices and Systems*. 5th ed. John Wiley, 1992 (see p. 37).

[4] W. Stallings. "Chapter 11: Digital logic". In: *Computer Organisation and Architecture*. 9th ed. Prentice-Hall, 2013 (see p. 37).

[5] A.S. Tanenbaum and T. Austin. "Section 3.1: Gates and Boolean algebra". In: *Structured Computer Organisation*. 6th ed. Prentice-Hall, 2012 (see p. 37).

[6] A.S. Tanenbaum and T. Austin. "Section 3.2.1: Integrated circuits". In: *Structured Computer Organisation*. 6th ed. Prentice-Hall, 2012 (see p. 37).

Notes: