1 Organization and Introduction

- The Art of managing complexity
  - Abstraction: Hiding details when they are not important
  - Discipline: Intentionally restricting your design choices so that you can work more productively at higher abstraction levels
  - The three -Y's
    - Hierarchy: A system is divided into modules of smaller complexity
    - Modularity: Having well defined functions and interfaces
    - Regularity: Encouraging uniformity, so modules can be easily re-used
- Bit: **B**inary dig**it**

2 Binary Numbers

- Powers of two:

$2^0 = 1$	$2^5 = 32$	$2^{10} = 1024$
$2^1 = 2$	$2^6 = 64$	$2^{11} = 2048$
$2^2 = 4$	$2^7 = 128$	$2^{12} = 4096$
$2^3 = 8$	$2^8 = 256$	$2^{13} = 8192$
$2^4 = 16$	$2^9 = 512$	$2^{14} = 16384$

- Binary to decimal conversion

$$\begin{aligned} 10011_2 &= 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 1 \\ &= 16 \times 1 + 8 \times 0 + 4 \times 0 + 2 \times 1 + 1 \times 1 \\ &= 16 + 0 + 0 + 2 + 1 = 19_{10} \end{aligned}$$

- Convert decimal to binary (roughly). Example with 47_{10} to binary

$2^6 = 64$	is $64 \leq 47$?	no	0	do nothing
$2^5 = 32$	is $32 \leq 47$?	yes	1	47-32=15
$2^4 = 16$	is $16 \leq 15$?	no	0	do nothing
$2^3 = 8$	is $8 \leq 15$?	yes	1	15-8=7
$2^2 = 4$	is $4 \leq 7$?	yes	1	7-4=3
$2^1 = 2$	is $2 \leq 3$?	yes	1	3-2=1
$2^0 = 1$	is $1 \leq 1$?	yes	1	1-1=0; done!

$\Rightarrow 47_{10}$ to binary is 0101111_2

- Binary values and range
  - $N-$digit decimal number
    - How many values: 10^N
    - Range: $[0, 10^N - 1]$
    - Example (3-digit number): $10^3 = 1000$ possible values, range:$[0, 999]$
  - $N-$bit binary number
    - How many values: 2^N

    - Range: $[0, 2^N - 1]$
    - Example (3-digit number): $2^3 = 8$ possible values, range:$[0, 7] = [000_2 \text{ to } 111_2]$

- Hexadecimal (Base-16) Numbers

Decimal	Hexadecimal	Binary		Decimal	Hexadecimal	Binary
0	0	0000		8	8	1000
1	1	0001		9	9	1001
2	2	0010		10	A	1010
3	3	0011		11	B	1011
4	4	0100		12	C	1100
5	5	0101		13	D	1101
6	6	0110		14	E	1110
7	7	0111		15	F	1111

- Bits, Bytes, Nibbles...

$$\underbrace{1}_{\text{MSB}}\,001011\,\underbrace{0}_{\text{LSB}}\quad \overbrace{1001\,\underbrace{0110}_{\text{nibble}}}^{\text{Byte}}\quad \underbrace{\text{CE}}_{\text{MSB}}\,\text{BF9A}\,\underbrace{\text{D7}}_{\text{LSB}}$$

  Where MSB=Most significant Bit and LSB=Least significant Bit

- Addition in base two works exactly the same as in base 10, using carries

- Overflow
  - Digital systems operate on a fixed number of bits
  - Addition overflows when the result is too big to fit in the available number of bits

- Signed Binary Numbers
  - Sign/Magnitude Numbers
    - 1 sign bit, $N - 1$ magnitude bits
    - Sign bit is the most significant (left-most) bit
    - Example: 4-bit sign/mag repr. of ± 6:
      - $+6 = \mathbf{0}110$
      - $-6 = \mathbf{1}110$
    - Range of an $N-$bit sign/magnitude number: $\left[-\left(2^{N-1} - 1\right), 2^{N-1} - 1\right]$
    - Problems:
      - Addition doesn't work
      - Two representations of 0 (± 0): 1000 and 0000
      - Introduces complexity in the processor design
  - One's Complement Numbers

* A negative number is formed by reversing the bits of the positive number (MSB still indicates the sign of the integer)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		One's Compl.	Unsigned
0	0	0	0	0	0	0	0	=	0	0
0	0	0	0	0	0	0	1	=	1	1
0	0	0	0	0	0	1	0	=	2	2
...	...	...	...	...	...	...	...	...	...	...
0	1	1	1	1	1	1	1	=	127	127
1	0	0	0	0	0	0	0	=	-127	128
1	0	0	0	0	0	0	1	=	-126	129
...	...	...	...	...	...	...	...	...	...	...
1	1	1	1	1	1	0	1	=	-2	253
1	1	1	1	1	1	1	0	=	-1	254
1	1	1	1	1	1	1	1	=	-0	255

* Range of $n-$bit number: $[-2^{n-1}-1, 2^{n-1}-1]$, 8 bits:$[-127, 127]$
* Addition: Done using binary addition with end-around carry. If there is a carry out of the MSB of the sum, this bit must be added to the LSB of the sum

– Two's Complement Numbers

* Don't have same problems as sign/magnitude numbers:
  · addition works
  · Single representation for 0
* Has advantages over one's complement:
  · Has a single 0 representation
  · Eliminates the end-around carry operation required in one's complement addition.
* A negative number is formed by reversing the bits of the positive number (MSB still indicates the sign of the integer) and adding 1:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		Two's Compl.	Unsigned
0	0	0	0	0	0	0	0	=	0	0
0	0	0	0	0	0	0	1	=	1	1
0	0	0	0	0	0	1	0	=	2	2
...	...	...	...	...	...	...	...	...	...	...
0	1	1	1	1	1	1	1	=	127	127
1	0	0	0	0	0	0	0	=	-128	128
1	0	0	0	0	0	0	1	=	-127	129
...	...	...	...	...	...	...	...	...	...	...
1	1	1	1	1	1	0	1	=	-3	253
1	1	1	1	1	1	1	0	=	-2	254
1	1	1	1	1	1	1	1	=	-1	255

* Same as unsigned binary, but the most significant bit (MSB) has value of -2^{N-1}
  · Most positive 4-bit number: 0111
  · Most negative 4-bit number: 1000
* The most significant bit still indicates the sign (1=neg., 0=pos.)
* Range of an $N-$bit two's comp. number: $[-2^{N-1}, 2^{N-1}-1]$, 8 bits:$[-128, 127]$

- Increasing bit width (assume from N to M, with $M > N$):
  - Sign-extension
    - Sign bit is copied into MSB
    - Number value remains the same
    - Give correct result for two's compl. numbers
    - Example 1:
      - 4-bit representation of $3 = \mathbf{0}011$
      - 8-bit sign-extended value: $\mathbf{00000}011$
    - Example 2:
      - 4-bit representation of $-5 = \mathbf{1}011$
      - 8-bit sign-extended value: $\mathbf{11111}011$
  - Zero-extension
    - Zeros are copied into MSB
    - Value will change for negative numbers
    - Example 1:
      - 4-bit value: $0011_2 = 3_{10}$
      - 8-bit zero-extended value: $\mathbf{0000}0011_2 = 3_{10}$
    - Example 2:
      - 4-bit value: $1011_2 = -5_{10}$
      - 8-bit zero-extended value: $\mathbf{0000}1011_2 = 11_{\mathbf{10}}$

3 Short Introduction to Electrical Engineering (EE Perspective)

- The goal of circuit design is to optimize:
  - Area: Net circuit area is proportional to the cost of the device
  - Speed/Throughput: We want circuits that work faster, or do more
  - Power/Energy
    - Mobile devices need to work with a limited power supply
    - High performance devices dissipate more than $100\text{W}/cm^2$
  - Design time
    - Designers are expensive
    - The competition will not wait for you
- (Frank's) Principles for engineering
  - Good engineers are lazy: They do not want to work unnecessarily, be creative
  - They know how to ask the question "why"?: take nothing for granted
  - Engineering is not a religion: Use what works best for you
  - Keep it simple and stupid: Engineers' job is to manage complexity

- Building blocks for microchips
  - Conductors: Metals (Aluminium, Copper)
  - Insulators: Glass (SiO_2), Air
  - Semiconductors: Silicon (Si), Germanium (Ge)
- N-type Doping: Add extra electron (negatively charged), zone becomes negatively charged

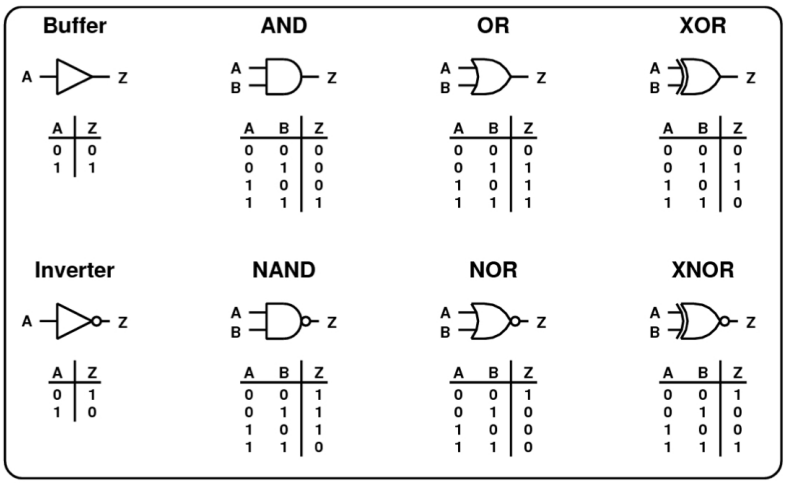
Maybe add a definition or a better explanation

- P-type Doping: Remove electron, zone becomes positively charged
- Semiconductors:
  - You can "Engineer" its properties, i.e.
    * Make it P type by injecting type-III elements (b, Ga, In)
    * Make it N type by injecting elements from type-V (P, As)
  - You can combine P and N regions to each other, from a pure semiconductor
  - Allows you to make interesting electrical devices (Diodes, Transistors, Thrystors)
- pMOS is a P type transistor, nMOS an N type transistors; combined they are a CMOS
- CMOS (Properties)
  - No input current: Capacitive input, no resistive path from the input
  - No current when output is at logic levels: Little static power, current is needed only when switching
  - Electrical properties determined directly by geometry: A transistor that is 2 times larger drives twice the current
  - Very simple to manufacture: pMOS and nMOS can be manufactures on the same substrate
- CMOS Gate Structure
  - The general form used to construct any inverting logic, such as: NOT, NAND, NOR
    * The networks may consist of transistors in series or parallel
    * When transistors are in parallel, the network is ON if either transistor is ON
    * When transistors are in series, the network is ON only if all transistors are ON
  - In a proper logic gate: One of the networks should be ON and the other OFF at any given time
  - Use the rule of conduction complements:
    * When nMOS transistors are in series, the pMOS transistor must be in parallel

    * When nMOS transistors are in parallel, the pMOS transistors must be in series

Add picture on slide 34, 03 - EEPerspective

- Logic Gates
  - Perform logic functions: Inversion (NOT), AND, OR, NAND, NOR, etc.
  - Single input: NOT gate, buffer
  - Two-input: AND, OR, XOR, NAND, NOR, XNOR



  - Multiple-Input:
    * 3, 4, or even more input AND, OR, XOR gates
    * Compound gates
      · AND-OR
      · OR-AND
      · AND-OR-INVERT
      · OR-AND-INVERT
    * Other cells: Multiplexers and Adders

- Logic Levels
  - Define ranges of discrete voltages to represent 1 and 0 (i.e. 0 for ground and 1 for 5V (V_{DD})) and allow for noise.

- Noise: Is anything that degrades the signal (i.e. resistance, power supply noise, etc.)

- Moore's Law
  - *"Number of transistors that can be manufactured doubles roughly every 18 months."* - Gordon Moore, 1965

- How do we keep Moore's Law:

- Manufacturing smaller structures: some structures are already a few atoms in size
- Developing materials with better properties
- Optimizing the manufacturing steps
- New technologies

- Power consumption
  - Power = Energy consumed per unit time
  - Two types of power consumption:
    1. Dynamic power consumption: Power to charge transistor gate capacitances
    $$P_{\text{dynamic}} = \frac{1}{2}CV_{DD}^2 f$$
    2. Static power consumption: Power consumed when no gates are switching, caused by the leakage current
    $$P_{\text{static}} = I_{DD}V_{DD}$$

- Power Consumption example:
  - Estimate the power consumption of a wireless handheld computer
    * $V_{DD} = 1.2V$
    * $C = 20nF$
    * $f = 1GHz$
    * $I_{DD} = 20mA$

$$\begin{aligned} P_{\text{total}} &= P_{\text{dynamic}} + P_{\text{static}} \\ &= \frac{1}{2}CV_{DD}^2 f + I_{DD}V_{DD} \\ &= \frac{1}{2}(20nF)(1.2V)^2(1GHz) + (20mA)(1.2V) \\ &= 14.4W \end{aligned}$$

4 Combinational Circuits: Theory

- Circuit elements. A circuit consists of:
  - Inputs
  - Outputs
  - Nodes (wires): Connections between I/O and circuit elements. To count them, look at
    * Outputs of every circuit elements
    * Inputs to the entire circuit
  - Circuit elements

- Types of Logic Circuits

- Combinational Logic
  - Memoryless
  - Outputs determined by current values of inputs
  - In some books called Combinatorial Logic
- Sequential Logic
  - Has Memory
  - Outputs determined by previous and current values of inputs

- Rules of Combinational Composition
  - Every circuit element is itself combinational
  - Every node of the circuit is either
    - Designated as an input to the circuit
    - Connects to exactly one output terminal of a circuit element
  - The circuit contains no cyclic paths: Every path through the circuit visits each node at most once

- Boolean Equations[1]
  - Functional specifications of outputs in terms of inputs.

- Boolean Algebra
  - Set of axioms and theorems to simplify Boolean equations
  - Like regular algebra, but in some cases simpler because variables only have 1 or 0 as a value
  - Axioms and theorems obey the principles of duality:
    - Stay corrected if: ANDs and ORs interchanged and 0's and 1's interchanged
    - Example:

	Dual
$\overline{0} = 1$	$\overline{1} = 0$
$B \cdot \overline{B} = 0$	$B + \overline{B} = 1$

- Boolean Axioms

	Axiom		Dual	Name
$A1$	$B = 0$ if $B \neq 1$	$A1'$	$B = 1$ if $B \neq 0$	Binary Field
$A2$	$\overline{0} = 1$	$A2'$	$\overline{1} = 0$	NOT
$A3$	$0 \cdot 0 = 0$	$A3'$	$1 + 1 = 1$	AND/OR
$A4$	$1 \cdot 1 = 1$	$A4'$	$0 + 0 = 0$	AND/OR
$A5$	$0 \cdot 1 = 1 \cdot 0 = 0$	$A5'$	$1 + 0 = 0 + 1 = 1$	AND/OR

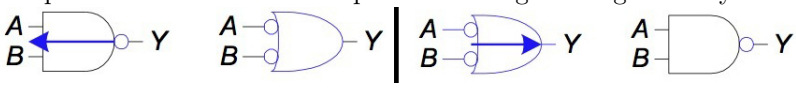
Duality: If the symbols 0 and 1 and the operators $\cdot$ (AND) and $+$ (OR) are interchanged, the statement will still be correct

[1] For a more in depth look, use the material from Diskrete Mathematik

- Boolean Theorems

	Theorem		Dual	Name
$T1$	$B \cdot 1 = B$	$T1'$	$B + 0 = B$	Identity
$T2$	$B \cdot 0 = 0$	$T2'$	$1 = 0$	Null Element
$T3$	$B \cdot B = B$	$T3'$	$1 + 1 = 1$	Idempotency
$T4$		$\overline{\overline{B}} = B$		Involution
$T5$	$B \cdot \overline{B} = 0$	$T5'$	$1 + 0 = 0 + 1 = 1$	Complements
$T6$	$B \cdot C = C \cdot B$	$T6'$	$B + C = C + B$	Commutativity
$T7$	$(B \cdot C) \cdot D = B \cdot (C \cdot D)$	$T7'$	$(B + C) + D = B + (C + D)$	Associtivity
$T8$	$(B \cdot C) + (B \cdot D) = B \cdot (C + D)$	$T8'$	$(B + C) \cdot (B + D) = B + (C \cdot D)$	Distributivity
$T9$	$B \cdot (B + C) = B$	$T9'$	$B + (B \cdot C) = B$	Covering
$T10$	$(B \cdot C) + (B \cdot \overline{C}) = B$	$T10'$	$(B + C) \cdot (B + \overline{C}) = B$	Combining
$T11$	$(B \cdot C) + (\overline{B} \cdot D) + (C \cdot D)$ $= B \cdot C + \overline{B} \cdot D$	$T11'$	$(B + tC) \cdot (\overline{B} + D) \cdot (C + D)$ $= (B + C) \cdot (\overline{B} + D)$	Consensus
$T12$	$\overline{B_0 \cdot B_1 \cdot B_2 \cdot \ldots} = (\overline{B_0} + \overline{B_1} + \overline{B_2} + \ldots)$	$T12'$	$\overline{B_0 + B_1 + B_2 + \ldots} = (\overline{B_0} \cdot \overline{B_1} \cdot \overline{B_2} \cdot \ldots)$	De Morgan's Theorem

- Bubble Pushing
  - Pushing bubbles backward (from the output) or forward (from the inputs) changes the body of the gate from AND to OR or vice versa
    - Pushing a bubble from the output back to the inputs puts bubbles on all gate inputs
    - Pushing bubbles on all gate inputs forward toward the output puts a bubble on the output and changes the gate body



  - Rules:
    - Begin at the output of the circuit and work toward the inputs
    - Push any bubbles on the final output back toward the inputs
    - Draw each gate in a form so that bubbles cancel

5 Combinational Circuits Design

- Some Definitions:
  - Complement: variable with a bar over it $(\overline{A}, \overline{B}, \overline{C})$
  - Literal: variable or its complement $(A, \overline{A}, B, \overline{B}, C, \overline{C})$
  - Implicant: product (AND) of literals $(A \cdot B \cdot \overline{C})$
  - Minterm: product (AND) that includes all input variables $(A \cdot B \cdot \overline{C})$
  - Maxterm: sum (OR) that includes all input variables $(A + \overline{B} + \overline{C})$
- Sum-of-Products (SOP) Form

aggiungere tabella cerchiata ai valori Y=1

A	B	Y	minterm
0	0	0	$\overline{A}\overline{B}$
0	1	1	$\overline{A}B$
1	0	0	$A\overline{B}$
1	1	1	AB

$$Y = F(A, B) = (\overline{A} \cdot B) + (A \cdot B)$$

- All boolean equations can be written in SOP form
  - Each row in a truth table has a minterm
  - A minterm is a product (AND) of literals
  - Each minterm is TRUE for that row (and only that row)
- Formed by ORing the minterms for which the output is TRUE

- The Dual: Product-of-Sums (POS) Form

aggiungere tabella cechiata ai valori Y=0

A	B	Y	maxterm
0	0	0	$A + B$
0	1	1	$A + \overline{B}$
1	0	0	$\overline{A} + B$
1	1	1	$\overline{A} + \overline{B}$

$$Y = F(A, B) = (A + B) \cdot (\overline{A} + B)$$

- All Boolean equations can be written in POS form
  - Each row in a truth table has a maxterm
  - A minterm is a sum (OR) of literals
  - Each minterm is FALSE for that row (and only that row)
- Formed by ANDing the maxterms for which the output is FALSE

- Karnaugh Maps (K-Maps)
  - Boolean expressions can be minimized by combining terms
  - K-maps minimize equations graphically
  - Rules:
    - Special order for bit combinations: $\overleftarrow{00, 01, 11, 10}$ (only one bit changes to the next)
    - Every 1 in a K-map must be circled at least once
    - Each circle must span a power of 2 (2^0 included) squares in each direction
    - Each circle must be as large as possible
    - A circle may wrap around the edges of the K-map
    - A "Don't care" (X) is circled only if it helps minimize the equation

- Circuit schematics
  - Inputs: left (or top) side of a schematic
  - Outputs: right (or bottom) side of a schematic
  - Circuits should flow from left to right
  - Straight wires are better than wires with multiple corners
  - Wires always connect at a T junction

  - A dot where wires cross indicated a connection between the wires
  - Wires crossing without a dot make no connection

- Additional Logic Levels: X and Z
  - Contention: X
    * When a signal is being driven to 1 and 0 simultaneously
    * Not a real level, could be any value (1,0 or something in between)
    * Usually a problem:
      · Two outputs drive one node to opposite values
      · Normally there should only be one driver for every connection
    * WARNING: "Don't care" and "contention" are both called X
      · These are not the same
      · Verilog uses X for both, VHDL uses "-" for don't care, and "X" for contention
      · Don't care: degree of freedom that is fixed at implementation time
      · Contention: a bug really, undetermined behaviour
  - High-impedance or tri-state (or Floating): Z
    * When an output is not driving to any specific value
    * Means the output is disconnected
    * Not a real level, some other output is able to determine the level
    * Output is called Floating, high impedance, tri-stated, or high-Z
    * Floating output might be 0, 1 or somewhere in between
    * Floating nodes are used in tri-state busses:
      · Many different drivers share one common connection
      · Exactly one driver is active at any time
      · All the other drivers are "disconnected"
      · The disconnected drivers are said to be floating, allowing exactly one node to drive
      · More than one input can listen to the shared bus without problems

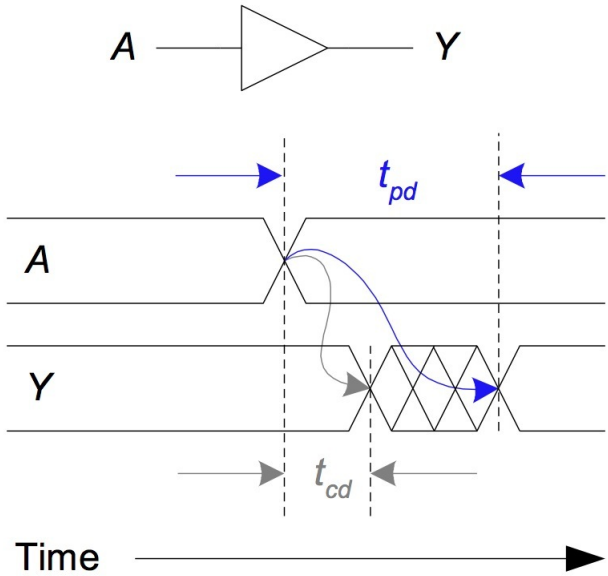
- Combinational Building Blocks
  - Combinational logic is often grouped into larger building blocks to build more complex systems
  - Hide the unnecessary gate-level details to emphasize the function of the building block (full adders, priority circuits, etc.)

- Multiplexer (Mux)
  - Selects between one of N inputs to connect to the output
  - Needs $\log_2 N-$bit control input
  - A 4:1 Multiplexer can be implemented with:

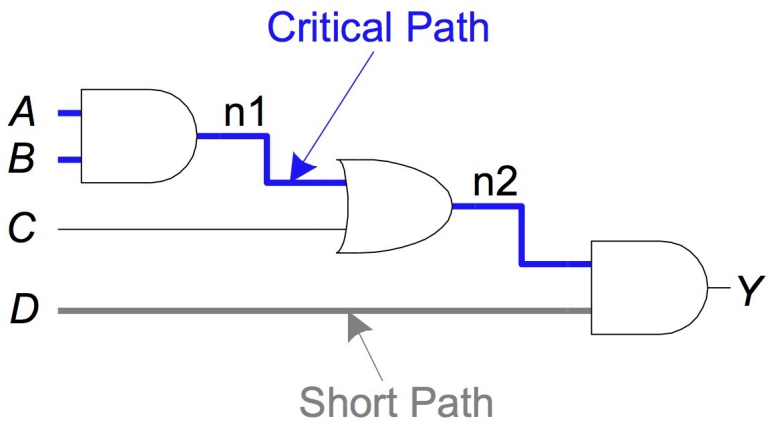
    * Two-level logic
    * Tristate buffers
    * Tree of 2:1 muxes
  - In general, a 2^N-input multiplexer can be programmed to perform any $N-$input logic function by applying 0's and 1's to the appropriate data inputs

- Decoders
  - N inputs, 2^N outputs
  - One-hot outputs: only one output HIGH at once

- Timing
  - Propagation delay: t_{pd} = max delay from input to output
  - Contamination delay: t_{cd} = min delay from input to output



  - Delay is caused by
    * capacitance and resistance in a circuit
    * Speed of light limitation (not as fast as you think)
  - Reasons why t_{pd} and t_{cd} may be different:
    * Different rising and falling delays
    * Multiple inputs and outputs, some of which are faster than other
    * Circuits slow down when hot and speed up when cold
  - Critical (Long) and short paths



    * Critical (Long) path: $t_{pd} = 2t_{pd_\text{AND}} + t_{pd_\text{OR}}$
    * Short path: $t_{cd} = t_{cd_\text{AND}}$
  - Propagation Times

Gate	t_{pd}(ps)
NOT	30
2−input AND	60
3−input AND	80
4−input OR	90
tristate (A to Y)	50
tristate (enable to Y)	35

- Glitches
  - Glitch: when a single input change causes multiple output changes
  - Glitches don't cause problems because of synchronous design conventions
  - But it's important to recognize a glitch when you see one in timing diagrams
  - In general a glitch can occur when a change in a single variable crosses the boundary between two prime implicants in a $K-$map.
  - You **can't** get rid of all glitches - simultaneous transitions on multiple inputs can also cause glitches

6 Field Programmable Gate Array (FPGA)

- Logic arrays
  - Programmable logic arrays (PLAs)
    * AND array followed by OR array
    * Perform combinational logic only
    * Fixed internal connections
    * Composed of:
      · LUTs (LookUp Tables): perform combinational logic

        - Flip-flops: perform sequential functions
        - Multiplexers connect LUTs and flip-flops
  - Field programmable gate arrays (FPGAs)
    * Array of configurable logic blocks (CLBs)
    * Perform combinational and sequential logic
    * Programmable internal connections
    * Composed of:
      - CLBs (Configurable Logic Blocks): Perform logic

        too deeply nested

      - **LUT**s (**L**ook**U**p **T**ables): perform combinational logic
      - Flip-flops: performs sequential functions
      - Multiplexers: connect LUTs and flip-flops

        fino qua

      - IOBs (Input/Output Buffers): Interface with outside world
      - Programmable interconnection: connect CLBs and IOBs
    * Some FPGAs include other building blocks such as multipliers and RAMs

7 Verilog for Combinational Circuits

- Two hardware description languages
  - Verilog
    * Developed in 1984
    * Became an IEEE standard (1364) in 1995
    * More popular in US
  - VHDL
    * Developed in 1981
    * Became an IEEE standard (1076) in 1987
    * More popular in Europe

    We used Verilog
- Defining a module
  - A module is the main buliding block in Verilog
  - Need to declare:
    * Name of the module
    * Types of its connections (input, output)
    * Names of its connections

The following two codes are identical

```
module test ( a, b, y );
        input a;
        input b;
        output y;

endmodule
```

```
module test ( input a,
              input b,
              output y );

endmodule
```

- You can also define multi-bit busses
  - [`range_start : range_end`]

  Example

```
input  [31:0] a;   // a[31], a[30] .. a[0]
output [15:8] b1;  // b1[15], b1[14] .. b1[8]
output [7:0]  b2;  // b2[7], b2[6] .. b1[0]
input         clk; // single signal
```

- Basic Syntax
  - Verilog is case sensitive:
    * SomeName **and** somename **are not the same!**
  - Names cannot start with numbers:
    * 2good **is not a valid name**
  - White space is ignored
  - Comments
    * Single Line comment starts with //
    * Multiline comments start and end with /*...*/

    Remember:
    * Use a consistent naming style
    * Use MSB to LSB ordering for busses, e.g a[31:0]and not a[0:31]
    * Define one module per file, it makes managing your design hierarchy easier
    * Use a file name that equals your module name
- Two main HDL's styles
  - Structural
    * Describe how modules are interconnected
    * Each module contains other modules (instances) and interconnections between them
    * Describes a hierarchy
  - Behavioral
    * The module body contains functional description of the circuit
    * Contains logical and mathematical operators

quando togli il todo, il titolo dell'immagine e l'immagine vengono separati su due pagine

Structural HDL Example

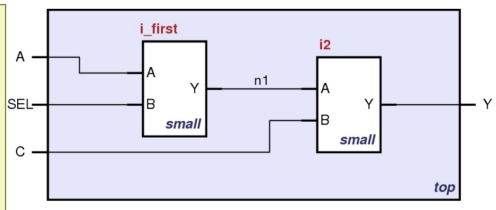
```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// alternative
small i_first ( A, SEL, n1 );

/* Shorter instantiation,
   pin order very important */

// any pin order, safer choice
small i2 ( .B(C),
           .Y(Y),
           .A(n1) );

endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

- Behavioral HDL: Defining Functionality

```
module example (a, b, c, y);
      input a;
      input b;
      input c;
      output y;

// here comes the circuit description
assign y = ~a & ~b & ~c |
           a & ~b & ~c |
           a & ~b &  c;

endmodule
```

- Bitwise Operators

```
module gates(input  [3:0]  a, b,
             output [3:0] y1, y2, y3, y4, y5);

   /* Five different two-input logic
      gates acting on 4 bit busses */

   assign y1 = a & b;     // AND
   assign y2 = a | b;     // OR
   assign y3 = a ^ b;     // XOR
   assign y4 = ~(a & b); // NAND
   assign y5 = ~(a | b); // NOR

endmodule
```

- Reduction Operators

```
module and8(input  [7:0] a,
            output       y);

   assign y = &a;

   // &a is much easier to write than
   // assign y = a[7] & a[6] & a[5] & a[4] &
   //            a[3] & a[2] & a[1] & a[0];

endmodule
```

- Conditional Assignments

```
module mux4(input  [3:0] d0, d1, d2, d3
            input  [1:0] s,
            output [3:0] y);

   assign y = s[1] ? ( s[0] ? d3 : d2)
                   : ( s[0] ? d1 : d0);
  // if (s1) then
  //      if (s0) then y=d3 else y=d2
  // else
  //      if (s0) then y=d1 else y=d0

endmodule
```

```
module mux4(input  [3:0] d0, d1, d2, d3
            input  [1:0] s,
            output [3:0] y);

   assign y = (s == 2'b11) ? d3 :
              (s == 2'b10) ? d2 :
              (s == 2'b01) ? d1 :
              d0;
// if      (s = "11" ) then y= d3
// else if (s = "10" ) then y= d2
// else if (s = "01" ) then y= d1
// else                     y= d0

endmodule
```

- How to express numbers

N'Bxx

8'b0000_0001

  - (N) Number of bits
    - Expresses how many bits will be used to store the value
  - (B) Base
    - Can be n (binary), h (hexadecimal), d (decimal), o (octal)
  - (xx) Number
    - The value expressed in base, aparto from numbers it can also have X and Z as values
    - Underscore_ can be used to improve readability

- Number representation in Verilog

Verilog	Stored Number
$4'b1001$	1001
$8'b1001$	0000 1001
$8'b0000_1001$	000 1001
$8'bxX0X1zZ1$	$XX0X$ $1ZZ1$
$'b01$	0000...0001
$4'd5$	0101
$12'hFA3$	1111 1001 0011
$8'o12$	00 001 010
$4'h7$	0111
$12'h0$	0000 0000 0000

non so cosa ce di importante da aggiungere...