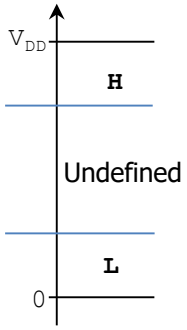
Unit 3 – Implementation Technology

FUNDAMENTALS

LOGIC LEVELS:



- Logic values are represented by TRUE or FALSE. In digital circuits, it is customary to represent them using voltage values: High voltage values are represented by H (or TRUE), and Low voltage values are represented by L (or FALSE).
- The value of V_{DD} depends on the I/O standard. For example: TTL (5v), LVTTL (3.3v), LVCMOS (2.5), LVCMOS (1.2).
- As for the symbols '1' and '0', there are two ways to assign them:
  - ✓ **Positive Logic**: The symbol '1' is assigned to H. The symbol '0' is assigned the L.
  - ✓ **Negative Logic**: The symbol '0' is assigned to H. The symbol '1' is assigned the L.

Example:

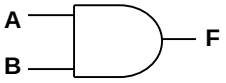
- A logic circuit has the following truth table. Using positive logic, the circuit represents an AND gate. Using negative logic, the circuit represents an OR gate. Small triangles are included in the gate's terminals to indicate we are using negative logic.

TRUTH TABLE OF GATE

A	B	F
L	L	L
L	H	L
H	L	L
H	H	H

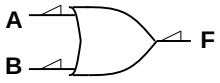
Positive Logic

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

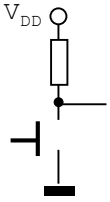


Negative Logic

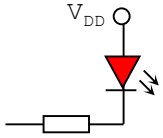
A	B	F
1	1	1
1	0	1
0	1	1
0	0	0



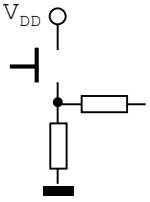
- Input and output interface pins can either be activated by a Low (L) or a High (H) value. Here, the distinction between positive and negative logic is important.



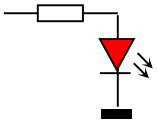
Active Low Switch



Active Low LED



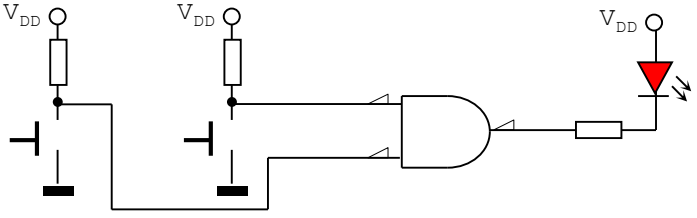
Active High Switch



Active High LED

Example:

- We want a circuit that activates an LED when the inputs are activated. But the inputs (switches) and output (LED) are active low. Here, negative logic seems better suited to represent this logic circuit: we need an AND gate in negative logic.



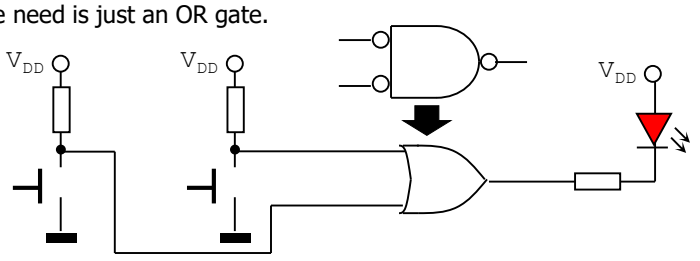
TRUTH TABLE OF GATE

A	B	F
L	L	L
L	H	H
H	L	H
H	H	H

Negative Logic

A	B	F
1	1	1
1	0	0
0	1	0
0	0	0

What about implementation? We use gates in positive logic: to do that, we replace the triangles with inverters, and finally what we need is just an OR gate.



TRUTH TABLE OF GATE

A	B	F
L	L	L
L	H	H
H	L	H
H	H	H

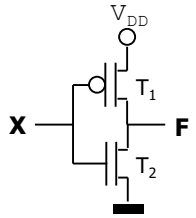
Positive Logic

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

- In conclusion, when encountering an active low input or output, include an inverter, and then just use positive logic. It is common to have a variety of input/output pins, some being active low and others being active high.

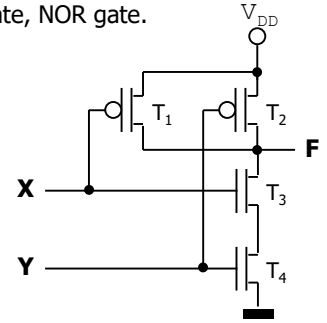
CMOS LOGIC GATES:

- CMOS (Complementary MOS): This type of gates includes PMOS and NMOS transistors.
- Example: NOT gate, NAND gate, NOR gate.



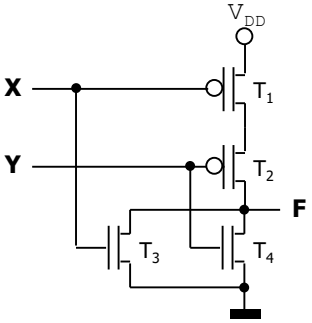
X	T_1	T_2	F
0	on	off	1
1	off	on	0

NOT gate



A	B	T_1	T_2	T_3	T_4	F
0	0	on	on	off	off	1
0	1	on	off	off	on	1
1	0	off	on	on	off	1
1	1	off	off	on	on	0

NAND gate



A	B	T_1	T_2	T_3	T_4	F
0	0	on	on	off	off	1
0	1	on	off	off	on	0
1	0	off	on	on	off	0
1	1	off	off	on	on	0

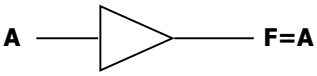
NOR gate

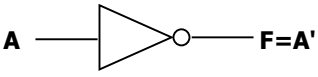
- We can implement a logic function using just transistors. This is more efficient resource-wise. However, for large designs, using logic gates is a better systematic way.

TRI-STATE BUFFERS:

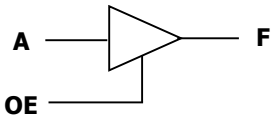
Buffers:

- They can drive more current (e.g.: motors, high-power LEDs) than simple logic gates. A common implementation uses OPAMPs.

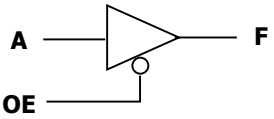




Tri-state Buffers:



OE = 0 → F = Z
OE = 1 → F = A

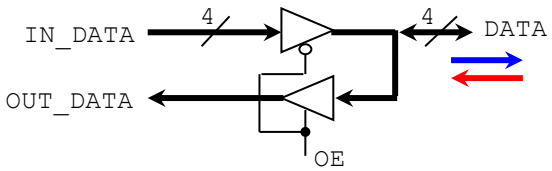


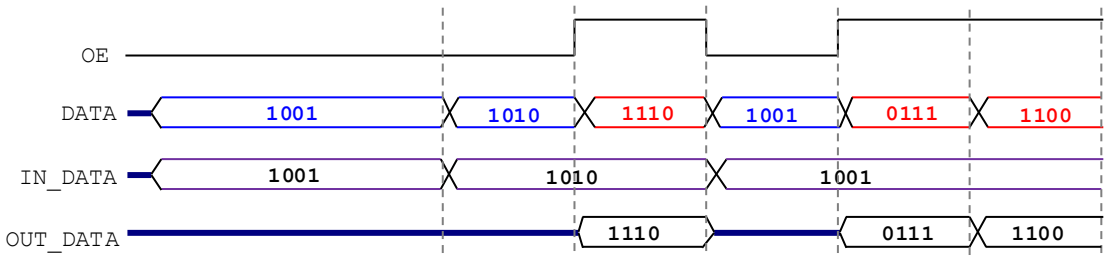
OE = 1 → F = Z
OE = 0 → F = A

- 'Z' State: This is high impedance, which effectively means that F is disconnected from A.

Applications:

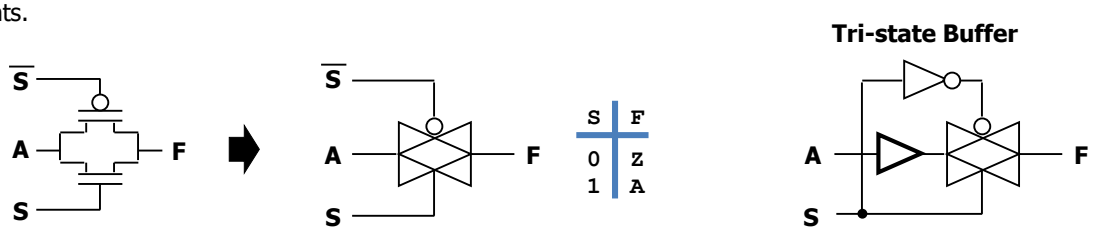
- Multiplexors, Bidirectional pins, Microprocessor Buses.
- Example: Bi-directional port (4 bits):





TRANSMISSION GATES

- This simple circuit behaves like a tri-state buffer. However, a tri-state buffer requires a NOT gate, and a buffer to drive high currents.



Applications:

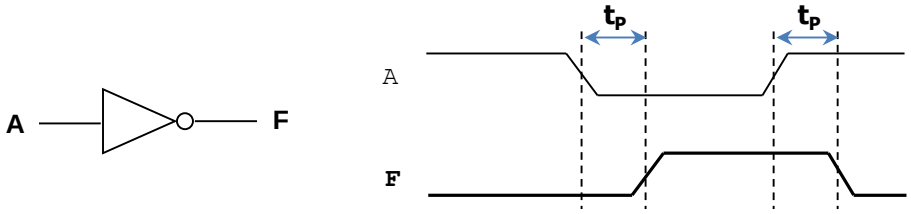
- MUXs, XOR gate.

PRACTICAL ASPECTS

- Digital circuits are analog circuits!

PROPAGATION DELAY:

- t_P: Propagation delay.

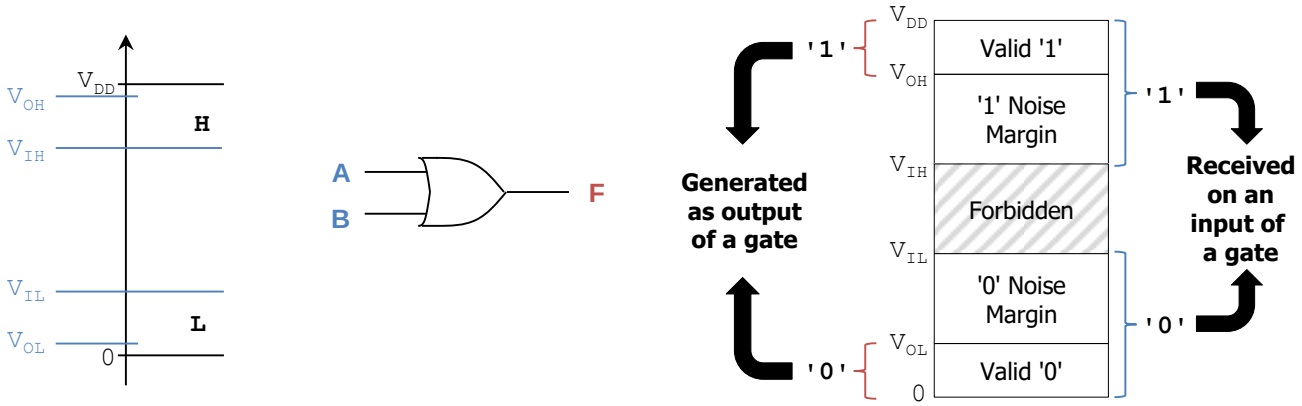


NOISE MARGIN

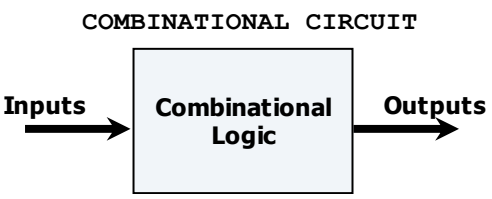
- Voltage definitions: The input voltages V_{IL}, V_{IH} are supposed to be generated by the output of other gates.
  - ✓ V_{OH}: Voltage produced by a gate when the output is HIGH.
  - ✓ V_{OL}: Voltage produced by a gate when the output is LOW.
  - ✓ V_{IL}: Maximum input voltage that the gate will interpret as LOW.
  - ✓ V_{IH}: Minimum input voltage that the gate will interpret as HIGH.

- In the figure below, we are using positive logic by assigning the logic value '1' to the high level value (H), and the logic value '0' to the low level value (L).

- **Noise margin**: Ability of the gate to tolerate noise. Electronic circuits are constantly subjected to random perturbation, called noise, which can alter the output voltage levels produced by a gate. This noise should not cause the receiving gate to misinterpret a low level value as a high one, a high as a low, or enter into a forbidden state.
  - ✓ Low noise margin: $NM_L = V_{IL} - V_{OL}$
  - ✓ High noise margin: $NM_H = V_{OH} - V_{IH}$

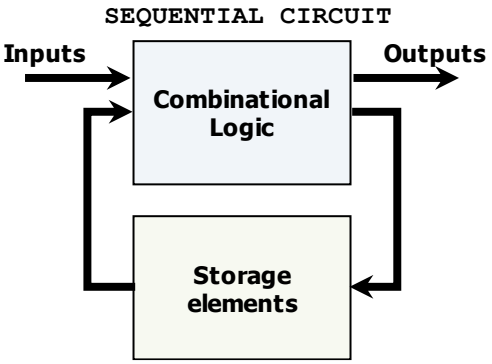


CLASSIFICATION OF DIGITAL CIRCUITS



- **Combinational circuits**: Here, the output values depend solely on the present input values. Circuit implementations of Boolean functions are combinational circuits. In Unit 5, we will present more complex combinational circuits that arise frequently in digital circuit design.

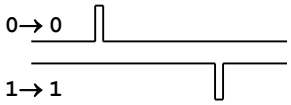
- **Sequential circuits**: Here, the output values depend on the present input values as well as on the previous values of the signals in the circuit (inputs, outputs, internal signals). Sequential circuits include storage elements to store those previous values. As a result, sequential circuits can implement a larger array of applications than combinational circuits. Unit 6 covers sequential circuits.

  The circuit state is defined as the contents of the storage elements at a given time. If the values of the inputs of the circuit change, the circuit might remain in the current state or it can change to a different one. A series of changes in the values of the inputs will cause the circuit to go through a sequence of states, hence the name sequential circuits. These circuits are built by interconnecting combinational circuits and storage elements. Sequential circuits are classified into:
  - ✓ *Asynchronous sequential circuits*: State changes depend upon the value of the inputs at any instant of time. They are difficult to design as we must take into account the propagation delay of the logic gates as well as the timing of the input changes.
  - ✓ *Synchronous sequential circuits*: State changes depend upon the value of the inputs at discrete instants of time. These circuits are simpler to design than asynchronous sequential circuits, and CAD (computer-aided-design) tool support is widespread. As such, these are the circuits most commonly encountered in practical applications.

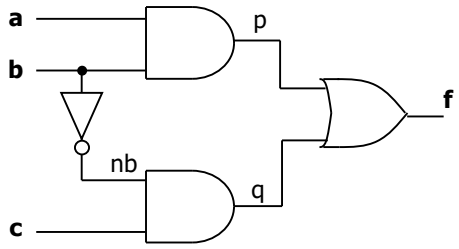
HAZARDS

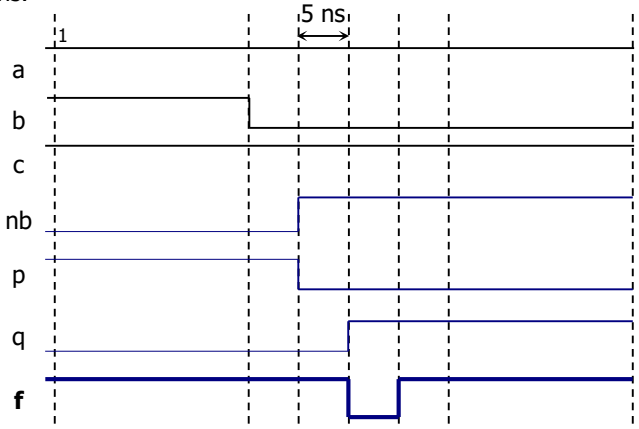
- A digital circuit can generate glitches, which are fast "spikes", usually unwanted.
- Glitches caused by the propagation delays and/or the structure of the circuit are known as hazards.
- Two types of hazard exist:
  - ✓ **Static Hazards**: They occur when the propagation delays are unbalanced. It can be addressed by adding all prime implicants to a function. These hazards happen when inputs change, but the output is not supposed to change. Two types: $0 \rightarrow 0$, or $1 \rightarrow 1$.

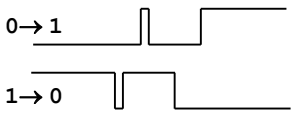
    Example: All gates have a propagation delay of 5 ns.

    $$f = ab + \bar{b}c$$

  - ✓ **Dynamic hazards**: They are caused by the structure of the circuit. They are difficult to detect and address. They usually occur in multilevel circuits. To avoid, use only two-level circuits and ensure that there are not static hazards. Two types: $1 \rightarrow 0$, or $0 \rightarrow 1$.

- **Significance of hazards**:
  - ✓ Combinational circuits: Hazards are usually not a problem because the outputs solely depend on the current inputs (as long as the duration between input changes is greater than the propagation delay, which is usually the case).
  - ✓ Asynchronous circuits: They are very vulnerable to hazards and will usually render the circuits unusable.
  - ✓ Synchronous circuits: Hazards do not pose a problem here, as we use registers to safely ignore hazards.

PROGRAMMABLE LOGIC DEVICES

- TTL (Transistor-Transistor logic) chips were commonly used to implement logic functions. Each one contains a few logic gates. The function of a TTL chip is fixed and cannot be modified to suit a particular design requirement. TTL chips are becoming obsolete for new designs.
- Programmable Logic Devices (PLDs) contain relatively large amounts of logic gates with a structure that is not fixed. A PLD is a general-purpose chip for implementing logic circuitry. A PLD contains logic gates and programmable switches. Once "programmed", PLDs maintain the configured circuitry, i.e., PLDs are non-volatile.
- PLDs are further classified into Simple PLDs (SPLDs) and Complex (CPLDs). They are suitable for relatively small to medium applications.
- A good overview is presented in: "FPGA and CPLD architectures: A Tutorial", S. Brown, J. Rose, *IEEE Design & Test of Computers*, vol. 13, no. 2, Summer 1996, pp. 42-57

SIMPLE PLDS (SPLDS):

- **Programmable Logic Array (PLA)**: It contains input buffers and inverters, an AND plane, and an OR plane. The AND and OR planes are configurable.
- **Programmable Array Logic (PAL)**: Similar to a PLA, but the OR plane is fixed. This is simpler to manufacture, but offers less flexibility.
- **Output pins of a PLA/PAL**: In order to support sequential circuits (synchronous, asynchronous), extra circuitry is added to the output pins. This extra circuitry commonly includes a flip flop, a multiplexor, a tri-state buffer and wires to feedback the output back to the AND plane. The extra circuitry along with the OR gates is called *macrocell*.

COMPLEX PLDS (CPLDS)

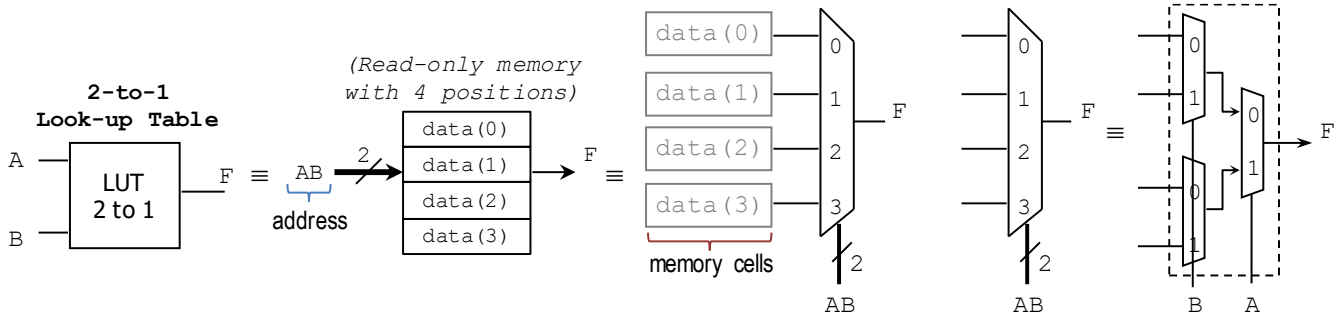
- They include multiple PAL-like blocks, with internal wiring resources to connect the circuit blocks. Each PAL-like block is also connected to subcircuit labeled I/O block.
- Commercial CPLDs range in size from 2 to 100 PAL-like blocks.
- Commercial CPLDs have an equivalent of up to 10,000 logic gates, which is not too large.
- Example of commercial CPLDs: Altera MAX7000, Xilinx Coolrunner-II.

FIELD PROGRAMMABLE GATE ARRAYS (FPGAS)

- FPGAs have large logic capacity and support the implementation of large logic circuits.
- They are volatile, so they have to be continuously power to maintain their configuration.
- They do not contain AND or OR planes. Instead, logic blocks are used to implement the required function.
- Three main types of resources: logic blocks, I/O blocks, and interconnection wires and switches.
- Example of commercial FPGAs: Xilinx Virtex-4/5/6, 7-series: Virtex-7, Artix-7, Kintex-7.

CONFIGURABLE LOGIC BLOCKS (CLBS):

- It contains LUTs (Look-up Tables) that implement the logic function. A LUT is also called a function generator, and it contains *storage cells* that can be programmed by a 1 or 0.
- **2-input LUT**: This circuit, depicted below, is equivalent to a 4-to-1 MUX with inputs being the contents of the memory cells. The contents of the storage cells are the minterms of the function $f(A,B)$. By modifying the contents of the memory cells, we can produce any 2-variable Boolean function.



The table below shows 3 functions that were generated by selecting particular values for the contents of the memory cells.

	A	***B***	Minterms	F1	F2	F3
0	0	0	$data(0) = m_0 = \bar{A}\,\bar{B}$	0	1	0
1	0	1	$data(1) = m_1 = \bar{A}B$	1	0	1
2	1	0	$data(2) = m_2 = A\bar{B}$	1	1	1
3	1	1	$data(3) = m_3 = AB$	0	1	1

- **n-input LUT**: We can build larger LUTs by using more MUXs. An n-input LUT can implement any n-variable Boolean function. Commercial devices usually implement LUTs of 4 to 6 inputs.
  - ✓ Xilinx Virtex-2/Virtex-4: 4-input LUTs
  - ✓ Xilinx Virtex-6/7-series: 6-input LUTs

- Each LUT is connected to a flip flop and a multiplexer.

PROGRAMMING PLDS, FPGAS

- In a PLD, the connection between logic signals and the gates in the AND/OR planes are specified by programmable switches. In an FPGA, the programmable switches allow the logic blocks to be interconnected.
- Commercial PLDs have a few thousand switches, and FPGA can have millions. Hence, it is not feasible for a user to specify manually the desired programmed state of each switch. Instead, CAD systems are employed: once the user completes the design of a circuit, the CAD tools generate a file, often called a *programming file* or *fuse map*, that specifies the state of each switch in the PLD for a given circuit. For an FPGA, the file is usually called the *bistream file*.
- PLDs/FPGAs have a dedicated programming unit, to which the programming file is streamed to.

Programmable Switches:

- Old technology: metal-alloys fuses were used as programmable links. Each pair of horizontal and vertical wires that cross is connected by a small metal fuse. To program, the fuse is melted for every connection that is not wanted on the circuit. The process is non-reversible.
- In current SPLDs, CPLDs, programmable switches are implemented by programmable transistors:
  - ✓ Erasable PROM (EPROM) transistor
  - ✓ Electrically erasable PROM (EEPROM) transistor
- In current FPGAs, the programming information is stored in memory cells, called static random access memory (SRAM) cells. SRAM cells are used to store data (truth table values) of the LUTs as well as the configuration of the interconnection wires on an FPGA. The collection of SRAM cells is called *Configuration Memory*. Due to the use SRAM technology, data is lost when the memory is not powered. Thus, the FPGA configuration is volatile.