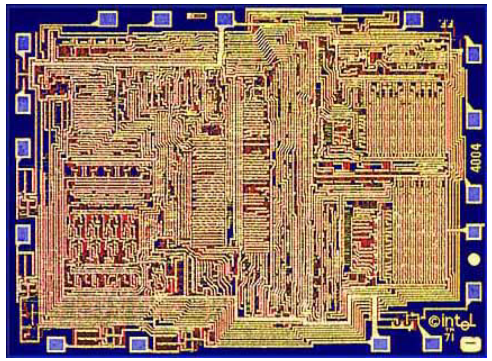
Basics of VLSI Design and IP

Prof. Maurizio MARTINA

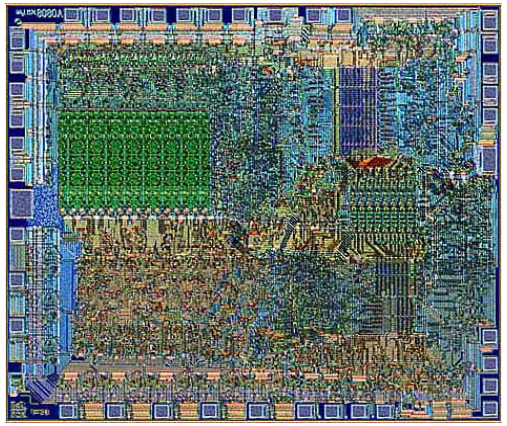
Introduction to VLSI Design

- VLSI Design: Very Large Scale Integration Design
  - Complex design
  - Large digital part
  - Massive production

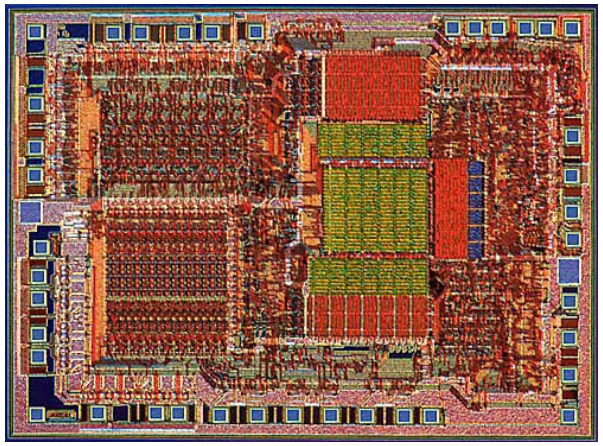
Evolution of VLSI technology



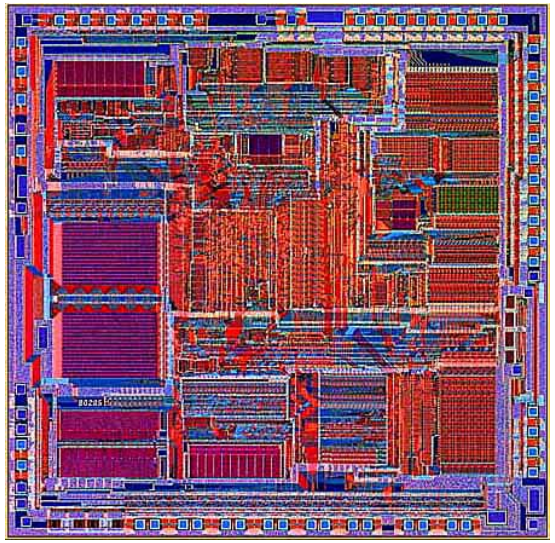
Intel 4004 (‘71)



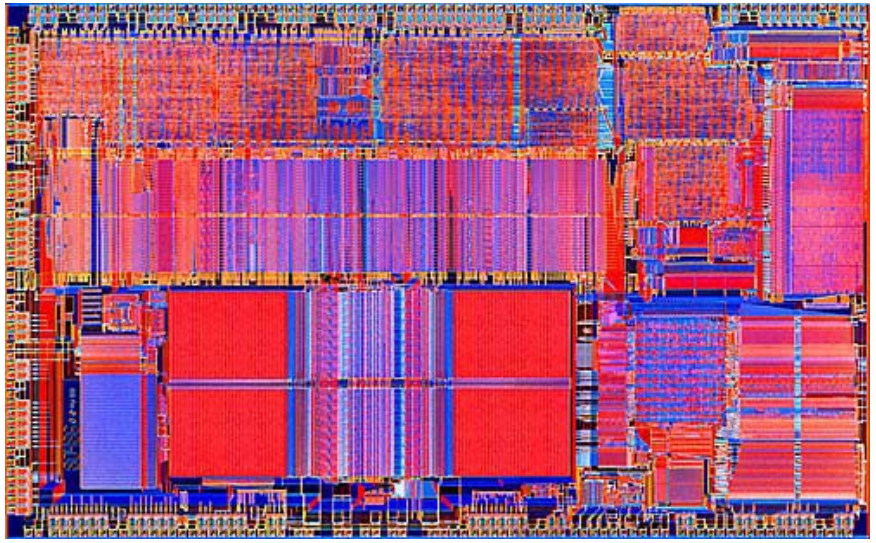
Intel 8080



Intel 8085

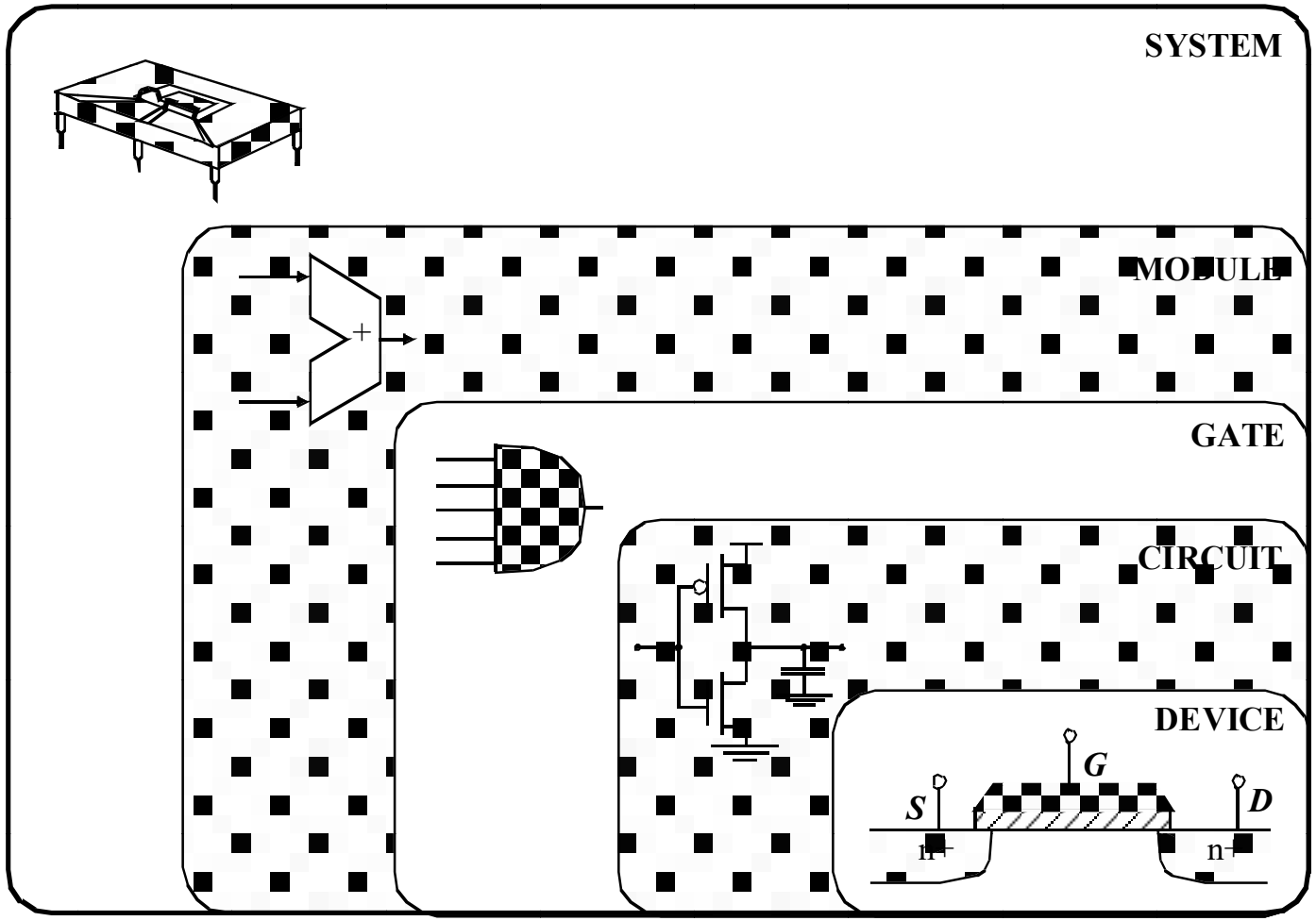


Intel 8286



Intel 8486

VLSI design process overview



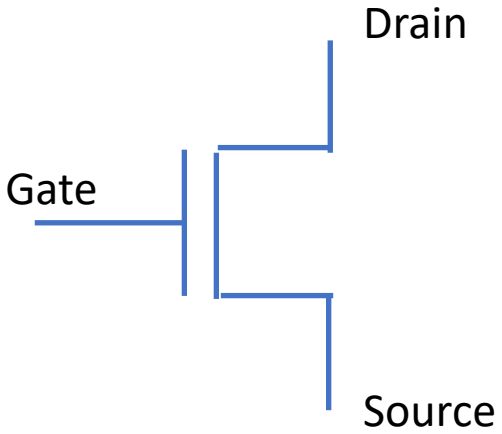
Register transfer level (RTL)

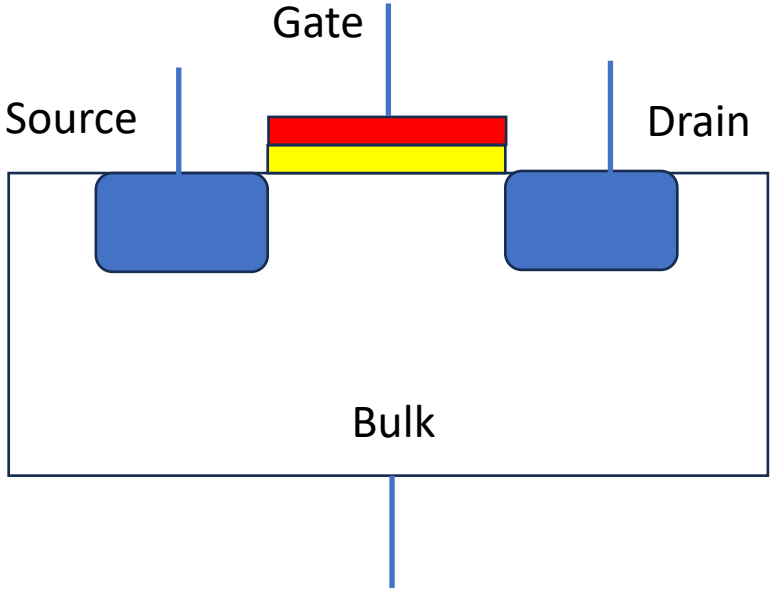
Layout level

Taken from «Digital Integrated Circuits», Rabaey, Chandrakasan, Nikolic

Device - I

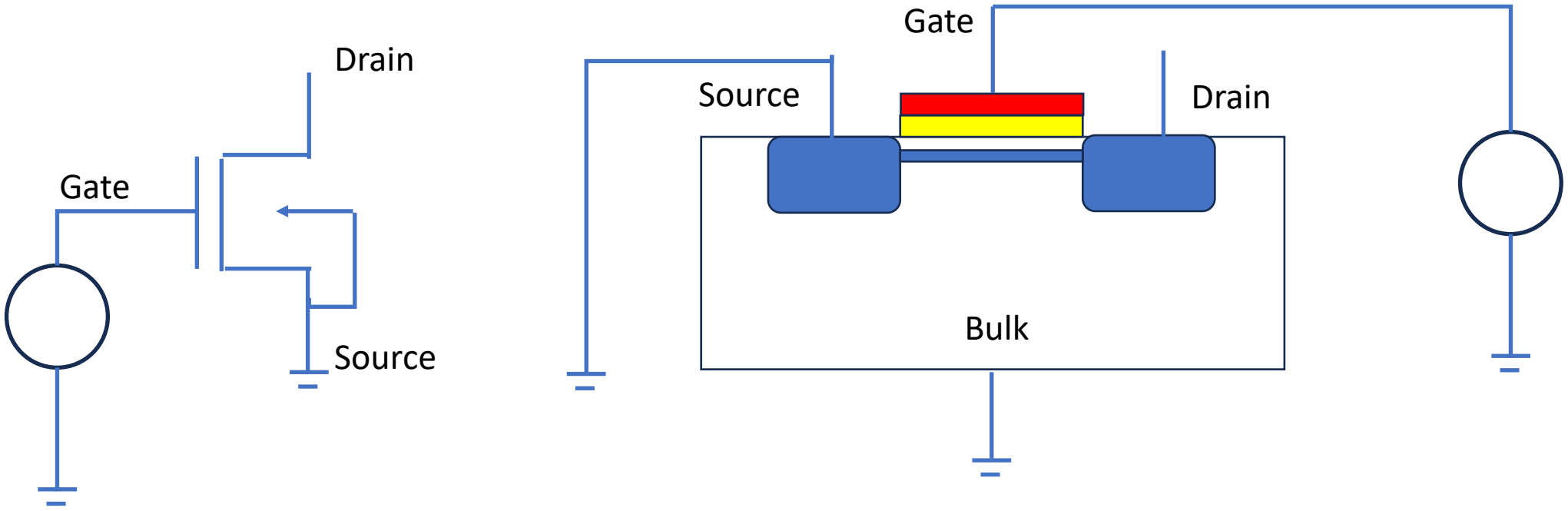
- MOS (Metal-Oxide-Semiconductor) Transistor model





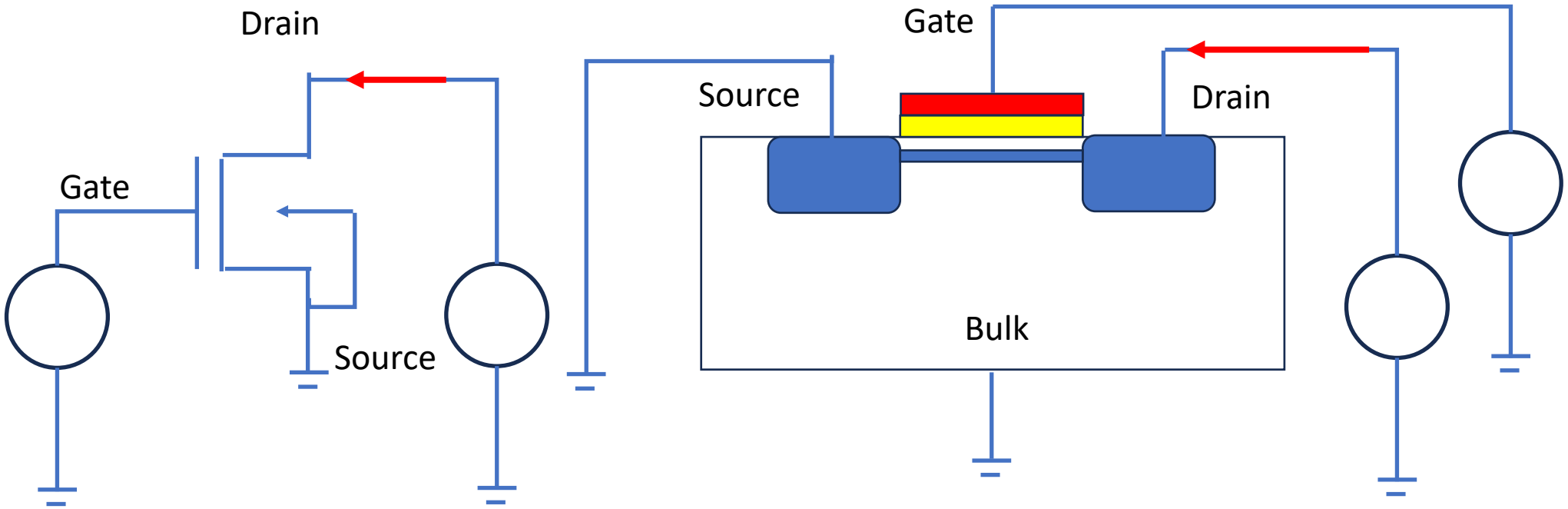
Device - II

- MOS (Metal-Oxide-Semiconductor) Transistor model



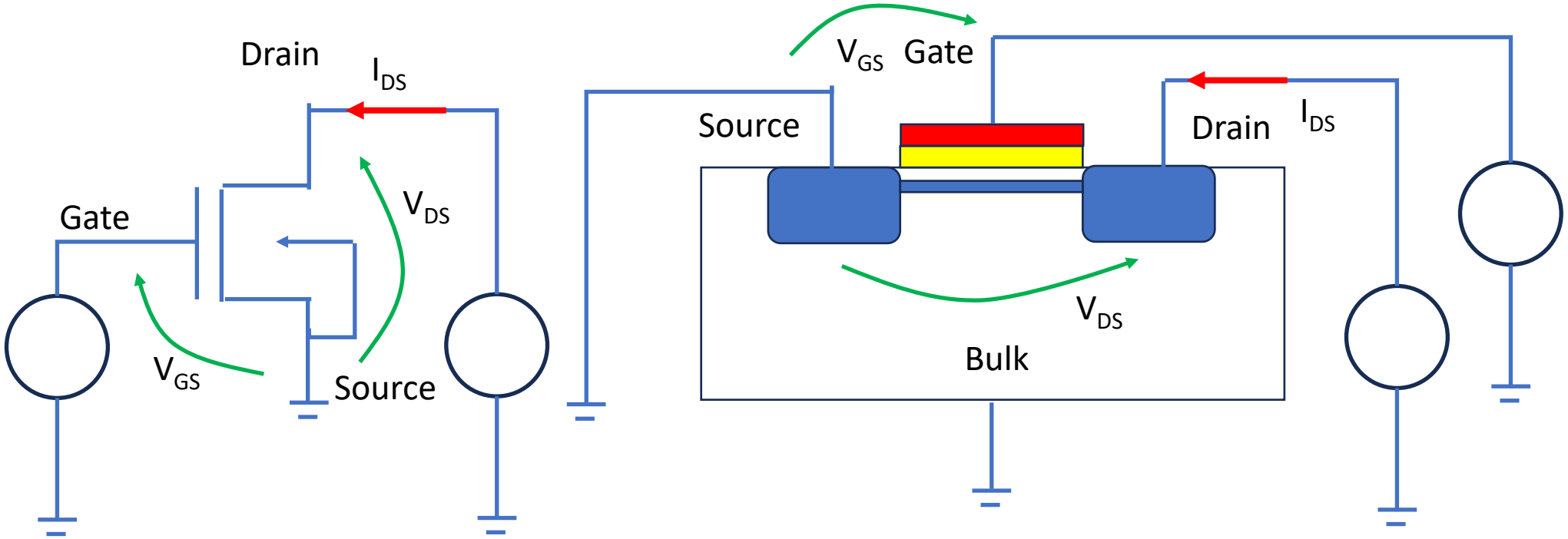
Device - III

- MOS (Metal-Oxide-Semiconductor) Transistor model



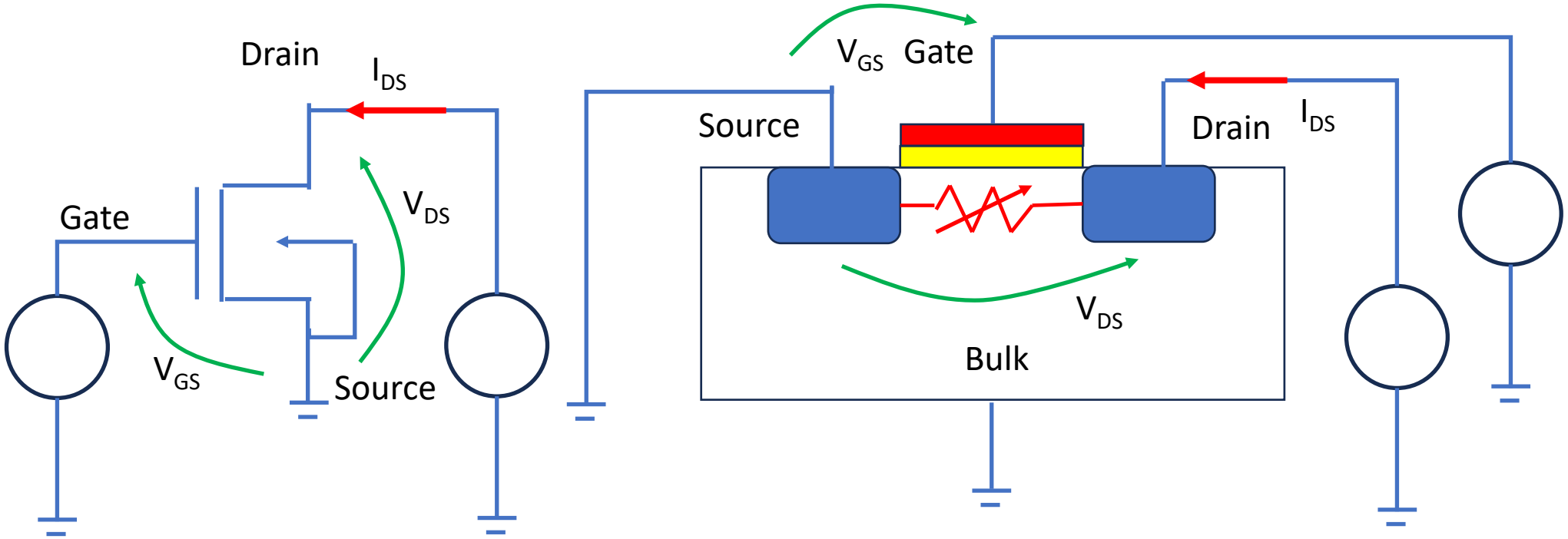
Device - IV

- MOS (Metal-Oxide-Semiconductor) Transistor model



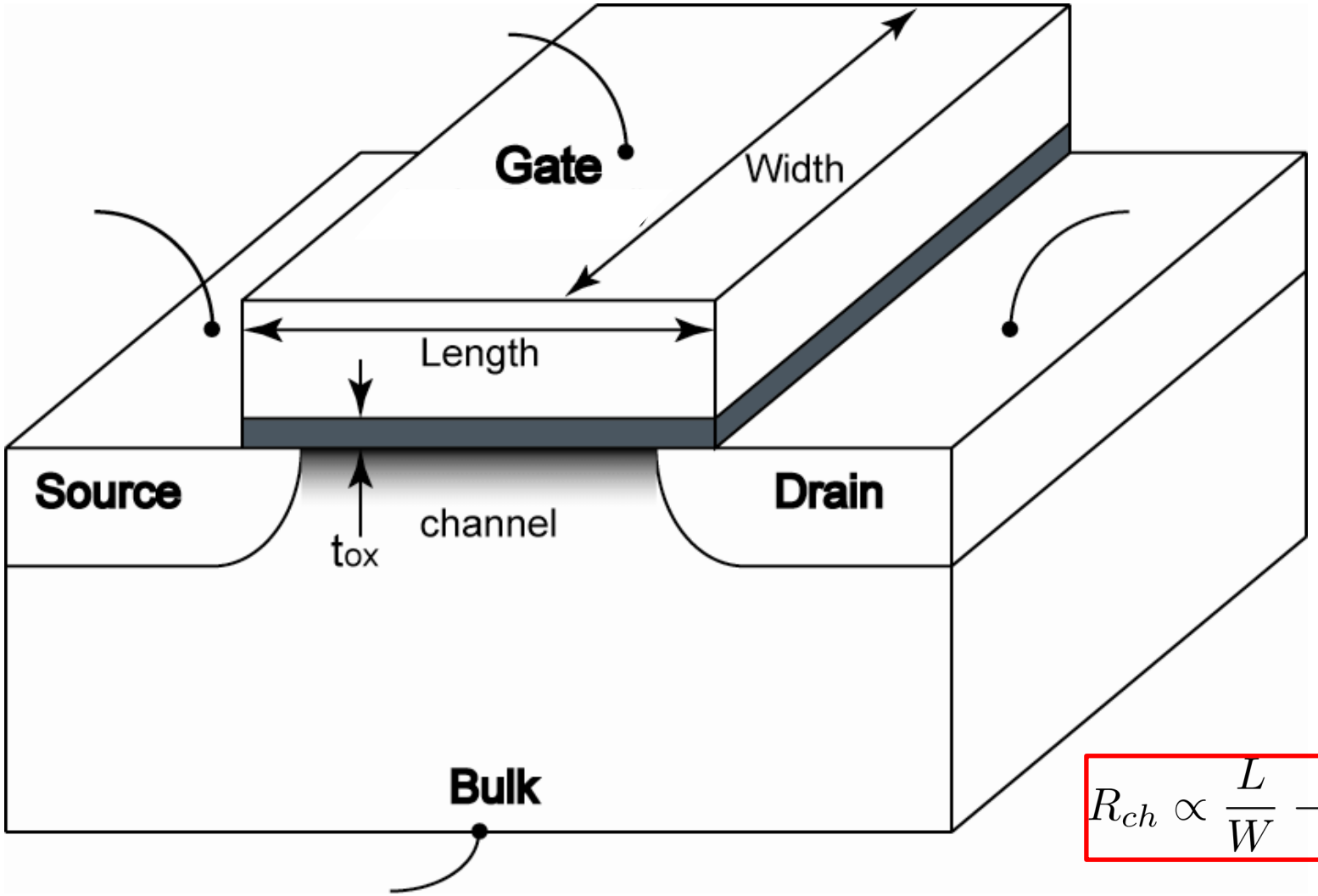
Device - V

- MOS (Metal-Oxide-Semiconductor) Transistor model



$$I_{DS} = f\left(V_{GS}, V_{DS}\right)$$

3D view



- The channel resistance depends also on geometrical parameters: Length (L) and Width (W)

$$R_{ch} \propto \frac{L}{W} \rightarrow I_{DS} \propto \frac{V_{DS}}{R_{ch}} \rightarrow I_{DS} \propto \frac{W}{L} V_{DS}$$

MOS transistor triode current

- It can be shown(*) that:

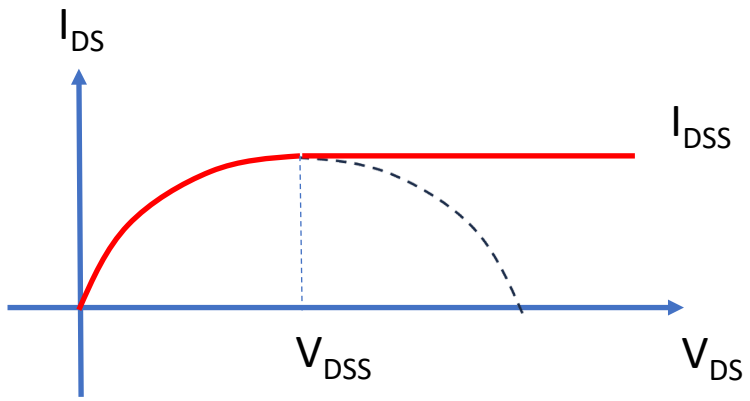
$$I_{DS} = \mu C_{ox} \frac{W}{L} \left[\left(V_{GS} - V_T \right) V_{DS} - \frac{V_{DS}^2}{2} \right]$$

- where we have parameters that depend on:
  - Technology
    - μ, C_{ox}, V_T → carriers mobility, oxide capacitance, threshold voltage
  - Circuit
    - V_{GS}, V_{DS}
  - Geometry
    - W, L

(*) R. Muller, T. Kamins "Device Electronics for Integrated Circuits”

Saturation

- The triode equation shows parabolic behaviour of I_{DS} as a function of V_{DS}
- Experimental results show that when the current reaches the maximum, it remains constant → saturation !



$$\frac{\partial I_{DS}}{\partial V_{DS}} = 0 \rightarrow k\left[(V_{GS} - V_T) - V_{DS}\right] = 0$$

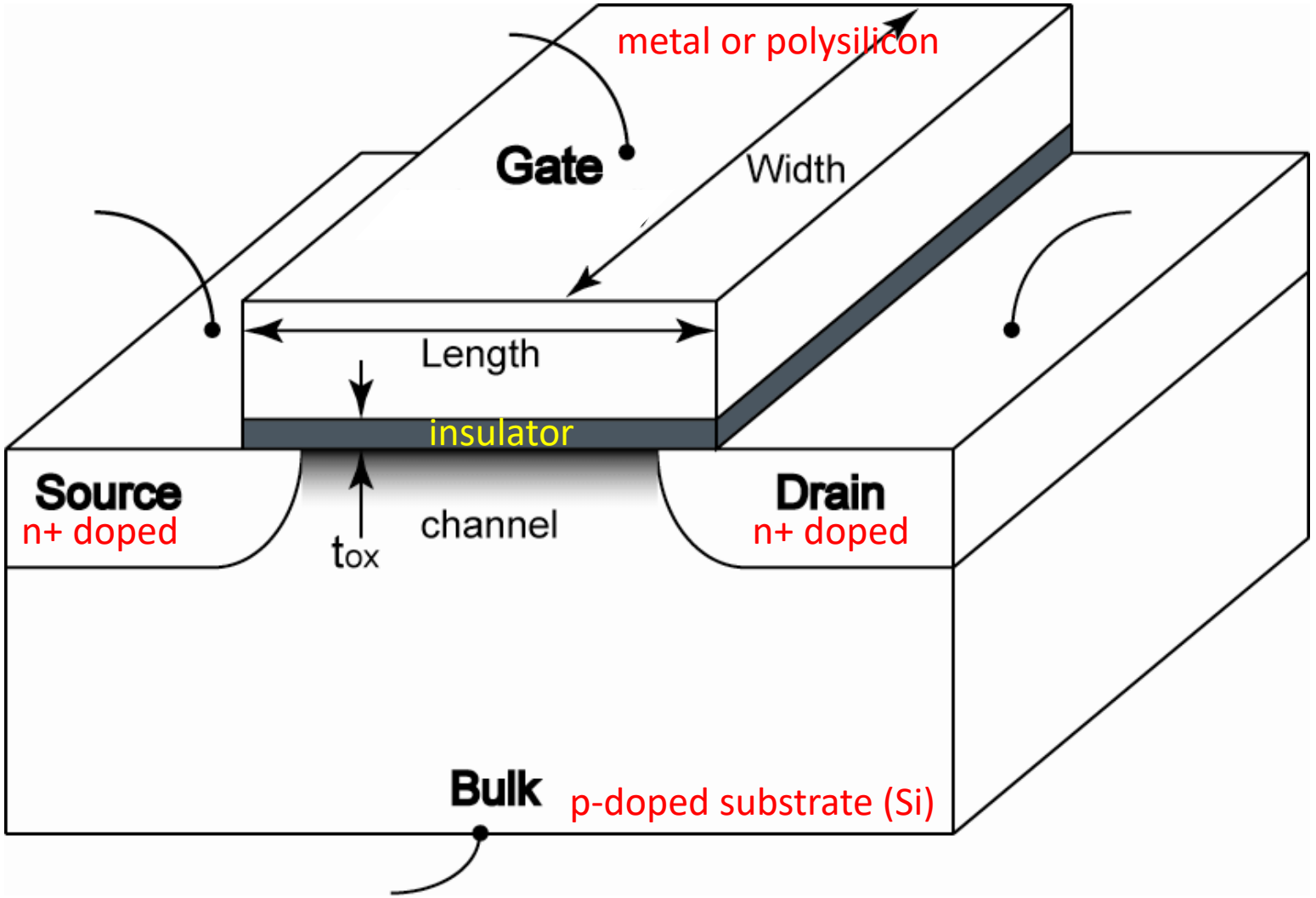
$$V_{DSS} = V_{GS} - V_T \rightarrow I_{DSS} = \frac{\mu C_{ox}}{2}\frac{W}{L}(V_{GS} - V_T)^2$$

Digital circuits

- V_{GS}, V_{DS} → low (gnd) or high (V_{dd})
  1. V_{GS} = low, V_{DS} = low → Transistor off → I_{DS}=0
  2. V_{GS} = low, V_{DS} = high → Transistor off → I_{DS}=0
  3. V_{GS} = high, V_{DS} = low → Transistor on, but I_{DS}≈0
  4. V_{GS} = high, V_{DS} = high → Saturation → I_{DS}=I_{DSS}

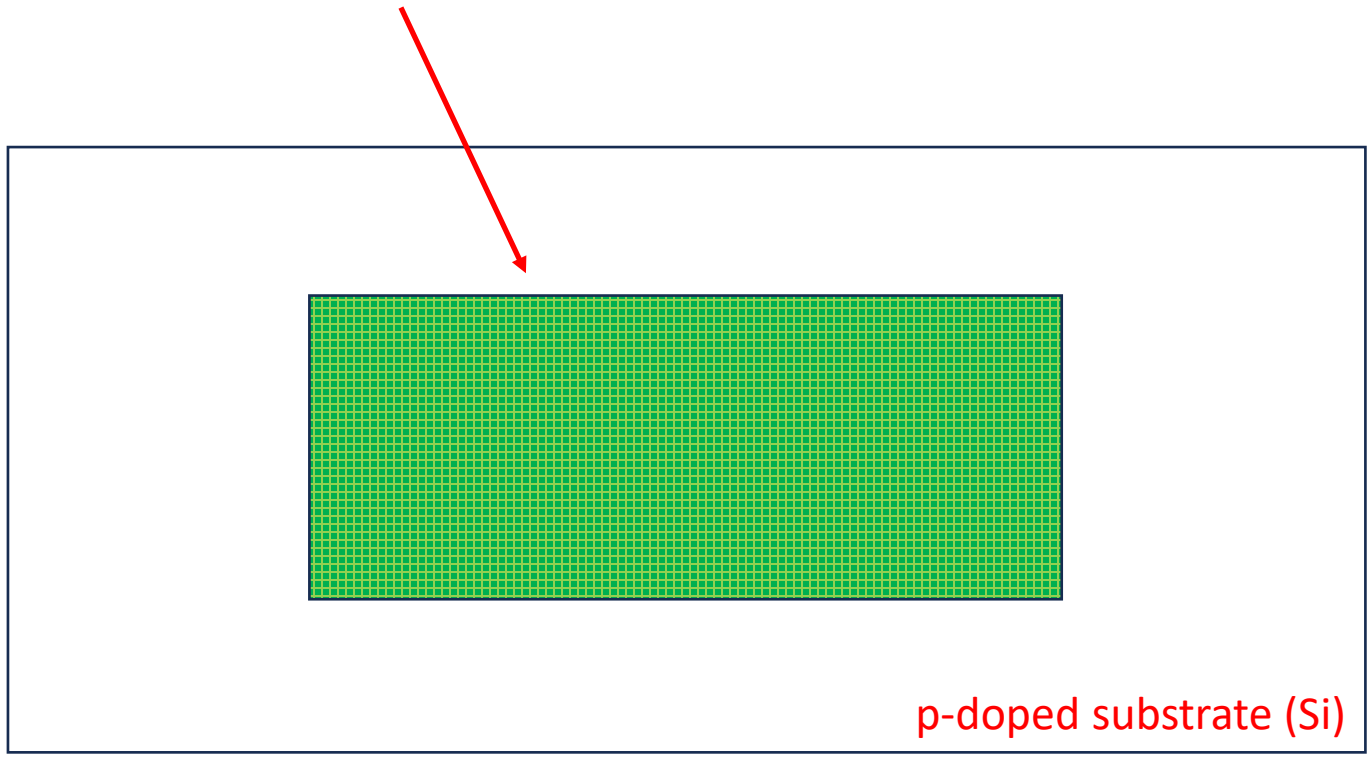
- Thus, given tecnology parameters, the designer can work on width (W) and lenght (L) of the MOS transistor.

Technology details: nMOS transistor



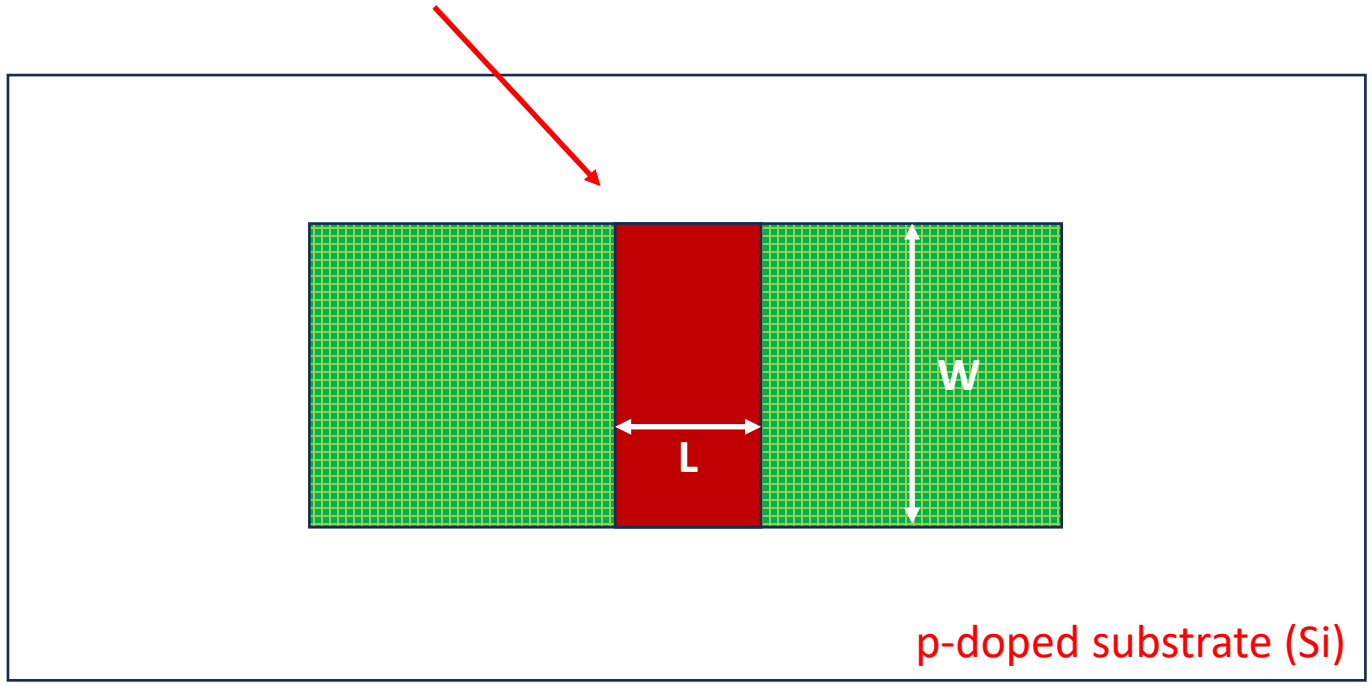
Technology steps - I

- Top view:
  - Define the n-region, where the nMOS is built.



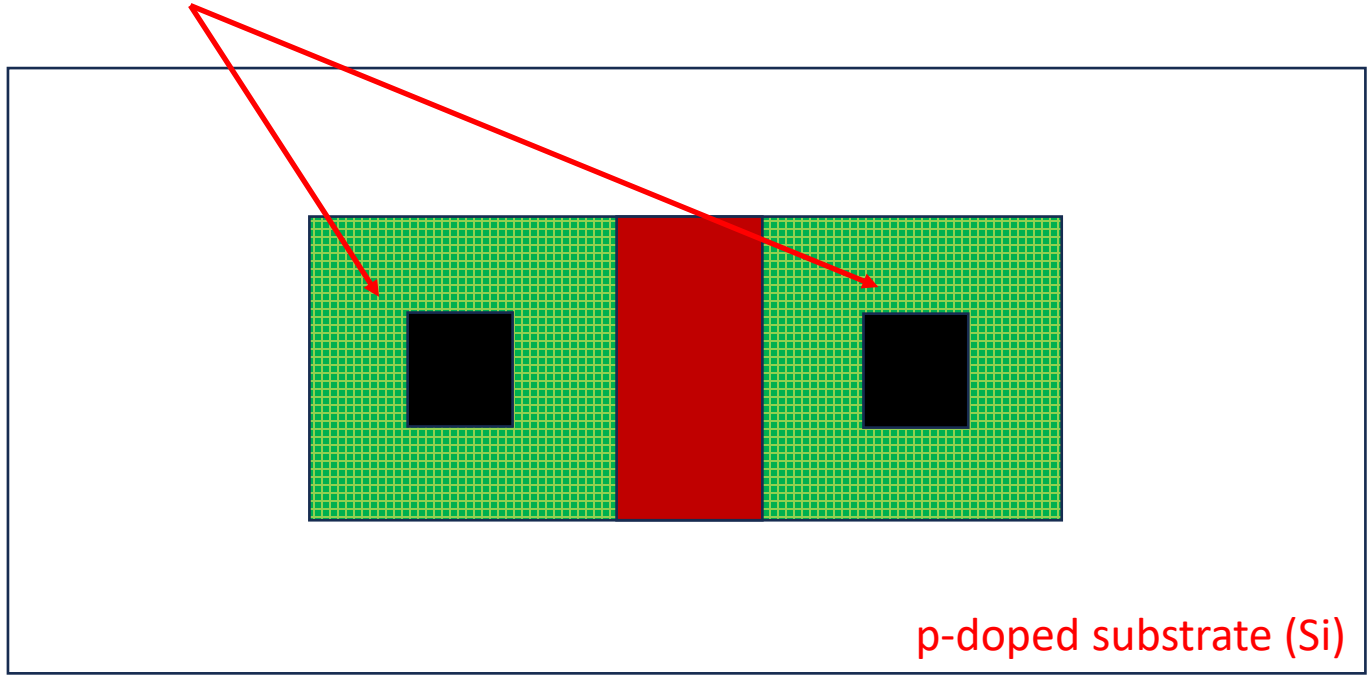
Technology steps - II

- Top view:
  - Define the n-region, where the nMOS is built.
  - Define the gate region of the nMOS



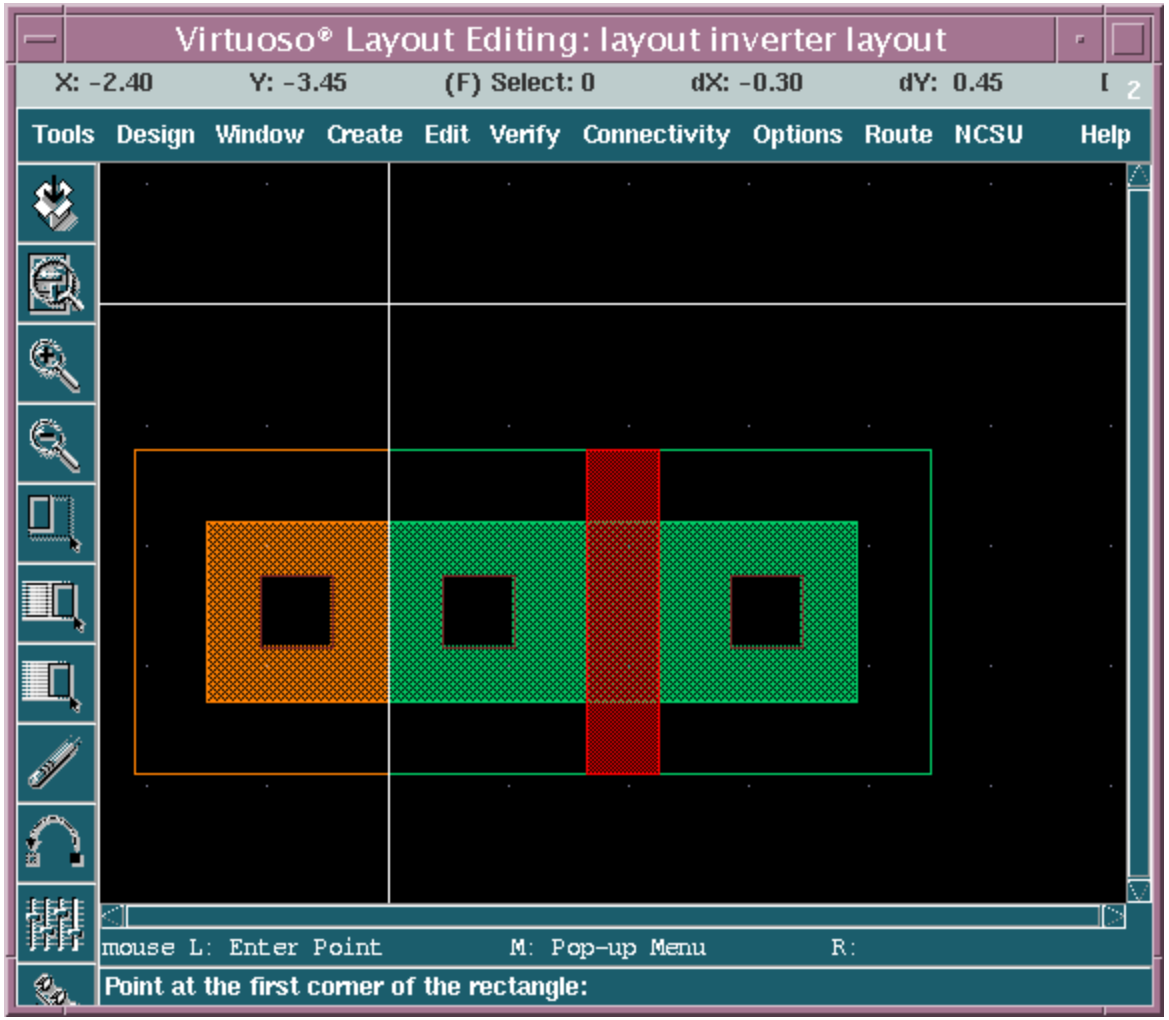
Technology steps - III

- Top view:
  - ...
  - Add source and drain metal contacts



Note: design rules

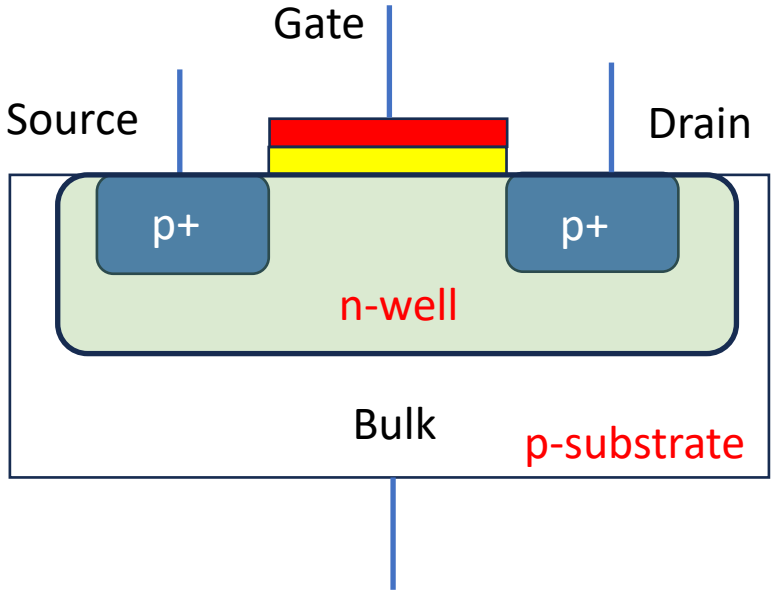
- Technology imposes some design rules:
  - Minimum feature size;
  - Minimum distance between different layers;
  - …
- Example: real nMOS layout with substrate contact (orange square)



Taken from Cadence tutorial, Prof. M.R. Stan, University of Virginia

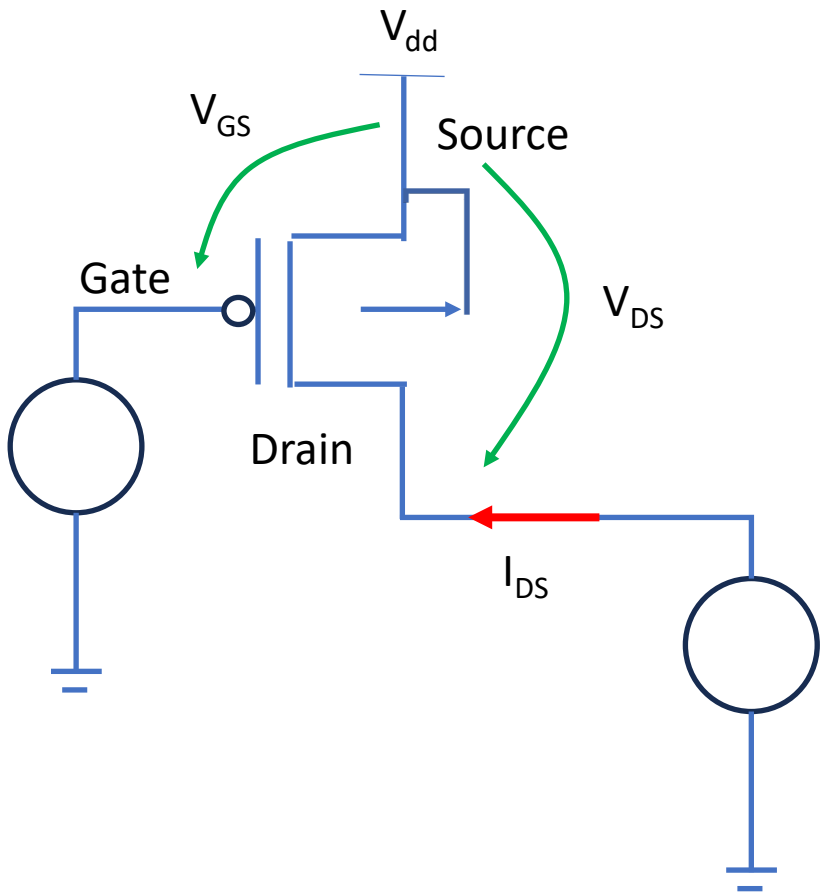
pMOS transistor - I

- Building a pMOS transistor on a p-doped substrate seems not possible.
- We need to build an n-doped well and then we can build a pMOS.



pMOS transistor - II

- It has a complementary behaviour with respect to nMOS transistors.



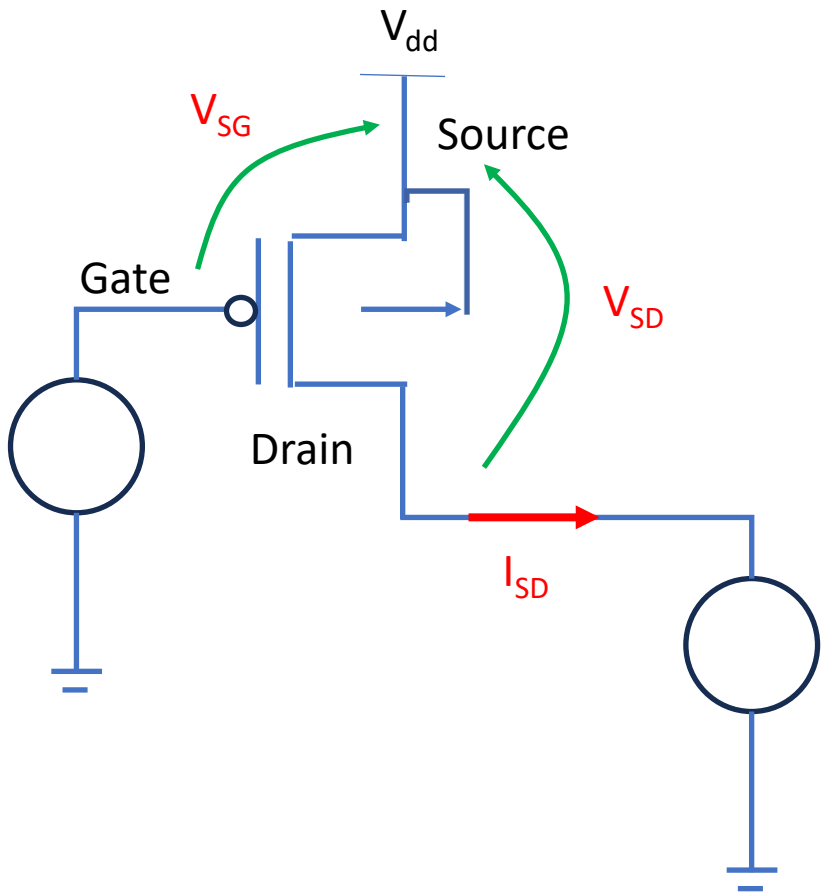
$$I_{DS} = \mu C_{ox} \frac{W}{L} \left[(V_{GS} - V_T) V_{DS} - \frac{V_{DS}^2}{2} \right]$$

< 0 !

$$I_{DS} = \begin{cases} 0 & V_{GS} > V_T \\ < 0 & V_{GS} \leq V_T \end{cases}$$

pMOS transistor - III

- It has a complementary behaviour with respect to nMOS transistors.

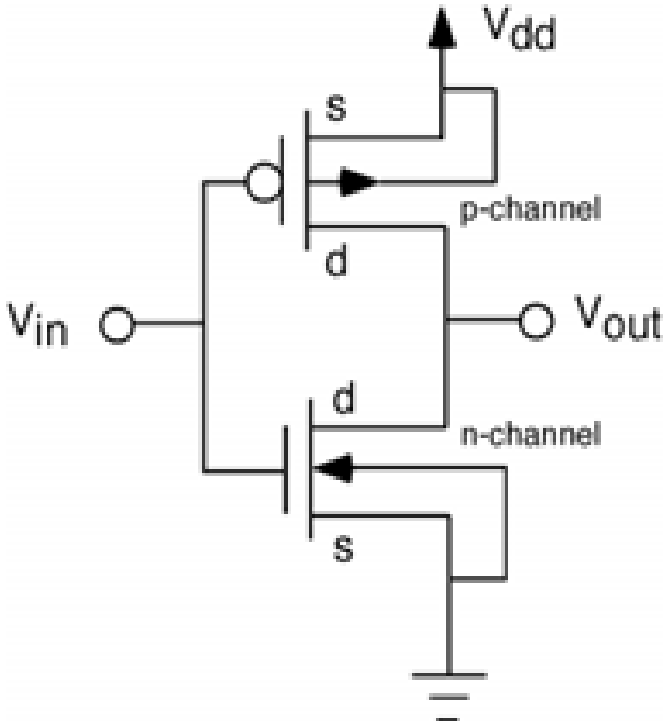


$$I_{SD} = \mu C_{ox} \frac{W}{L} \left[\left(V_{SG} - |V_T| \right) V_{SD} - \frac{V_{SD}^2}{2} \right]$$

$$I_{SD} = \begin{cases} 0 & V_{SG} < |V_T| \\ > 0 & V_{SG} \geq |V_T| \end{cases}$$

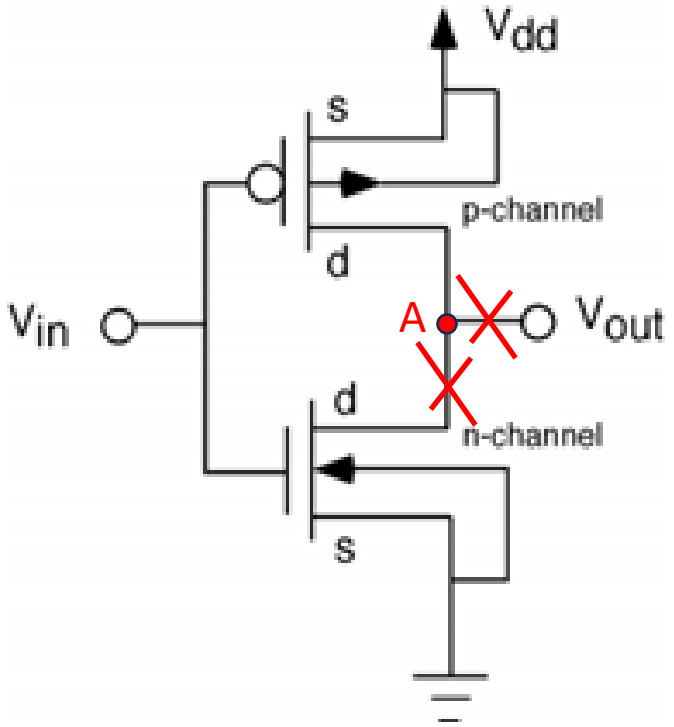
CMOS Inverter - I

- Use nMOS and pMOS transistors in one circuit
- Example: inverter



CMOS inverter - II

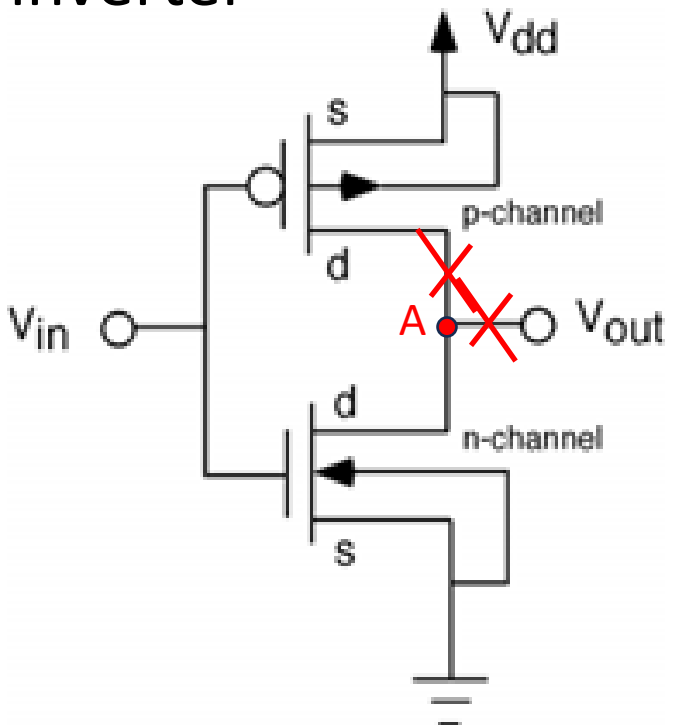
- Use nMOS and pMOS transistors in one circuit
- Example: inverter



1. If V_{in}=0 (gnd – logic ‘0’) the nMOS is off (I_{DS}=0), the pMOS is on;
2. KCL at node A says that the current in the pMOS must be zero too. So V_{SD} of the pMOS is zero and V_{out}=V_{dd} (logic ‘1’)

CMOS inverter - III

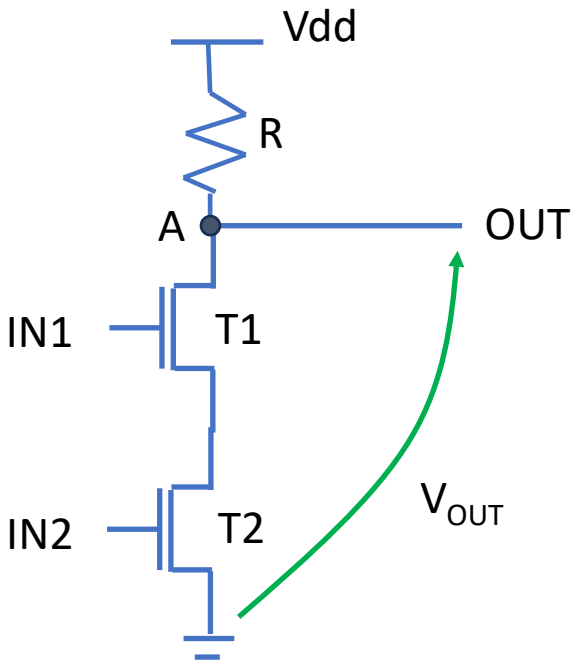
- Use nMOS and pMOS transistors in one circuit
- Example: inverter



1. If $V_{in}=V_{dd}$ (logic '1') the nMOS is on, the pMOS is off ($I_{SD}=0$);
2. KCL at node A says that the current in the nMOS must be zero too. So V_{DS} of the nMOS is zero and $V_{out}=0$ (logic '0')

Other logic functions - I

- Using pMOS and nMOS we can easily build NAND and NOR gates.
  - Example:

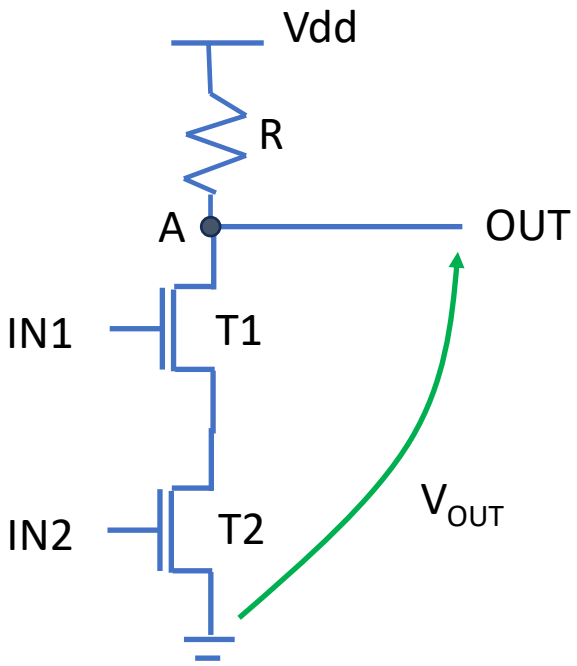


- If IN1='1' and IN2='1', then T1 and T2 are on.
- Thus, Vout ≈ 0 and so OUT='0'

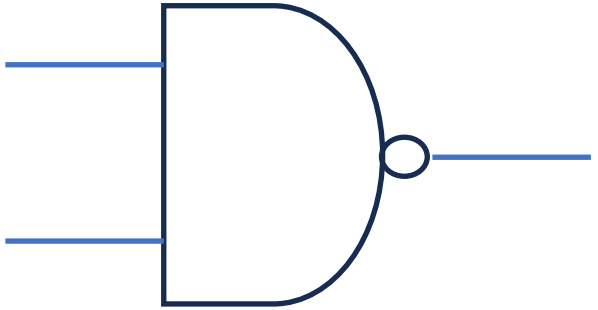
- If IN1='0' (or IN2='0'), then I_{DS1}=0 (or I_{DS2}=0). Thus, no current flows in the branch with T1 and T1
- KCL at node A says that no current flows in R, so no voltage drop across R.
- As a consequence, Vout=Vdd and OUT='1'

IN1	IN2	OUT
0	0	1
0	1	1
1	0	1
1	1	0

Other logic functions - II

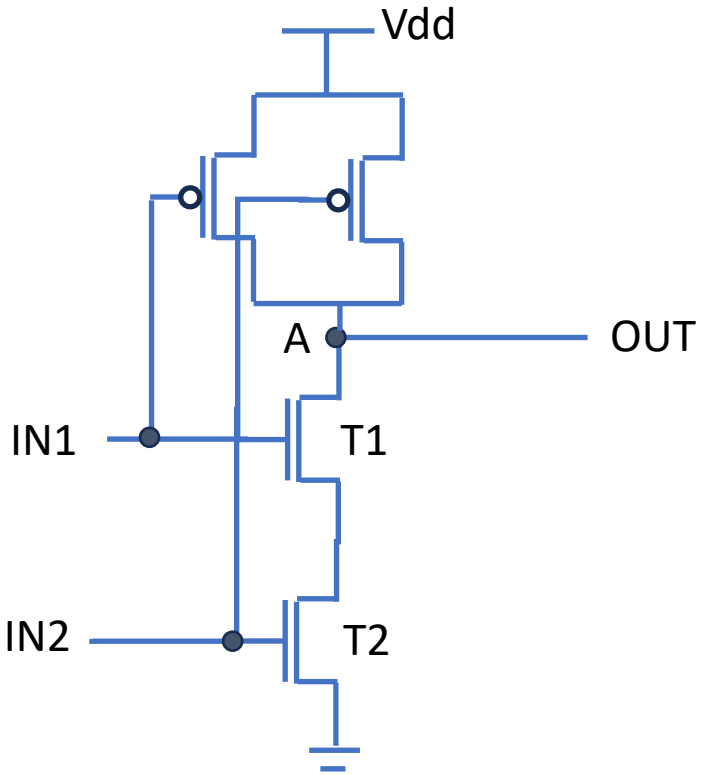


IN1	IN2	OUT
0	0	1
0	1	1
1	0	1
1	1	0

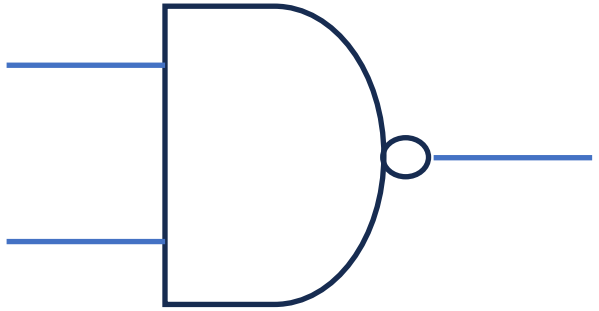


Other logic functions - III

- Exploiting pMOS/nMOS complementary behaviour, we obtain



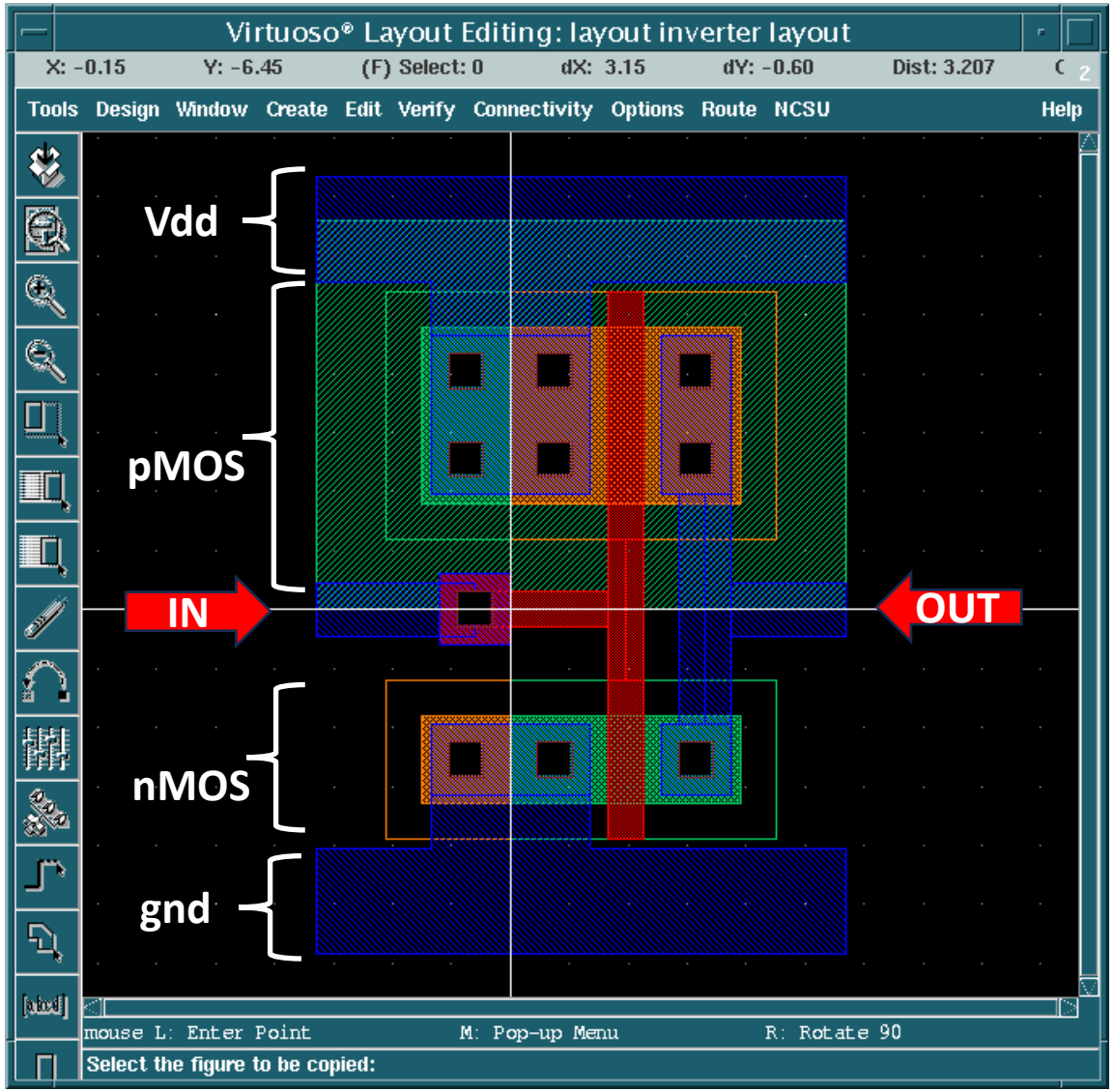
IN1	IN2	OUT
0	0	1
0	1	1
1	0	1
1	1	0



CMOS logic

- We can extend the NAND example to other gates (NOR, …)
- The nMOS branch and pMOS branch work in complementary mode to implement a Complementary MOS (CMOS) logic.
- Once we know the logic function we can design the layout of the corresponding circuit.

Example: CMOS inverter layout



Taken from Cadence tutorial, Prof. M.R. Stan, University of Virginia

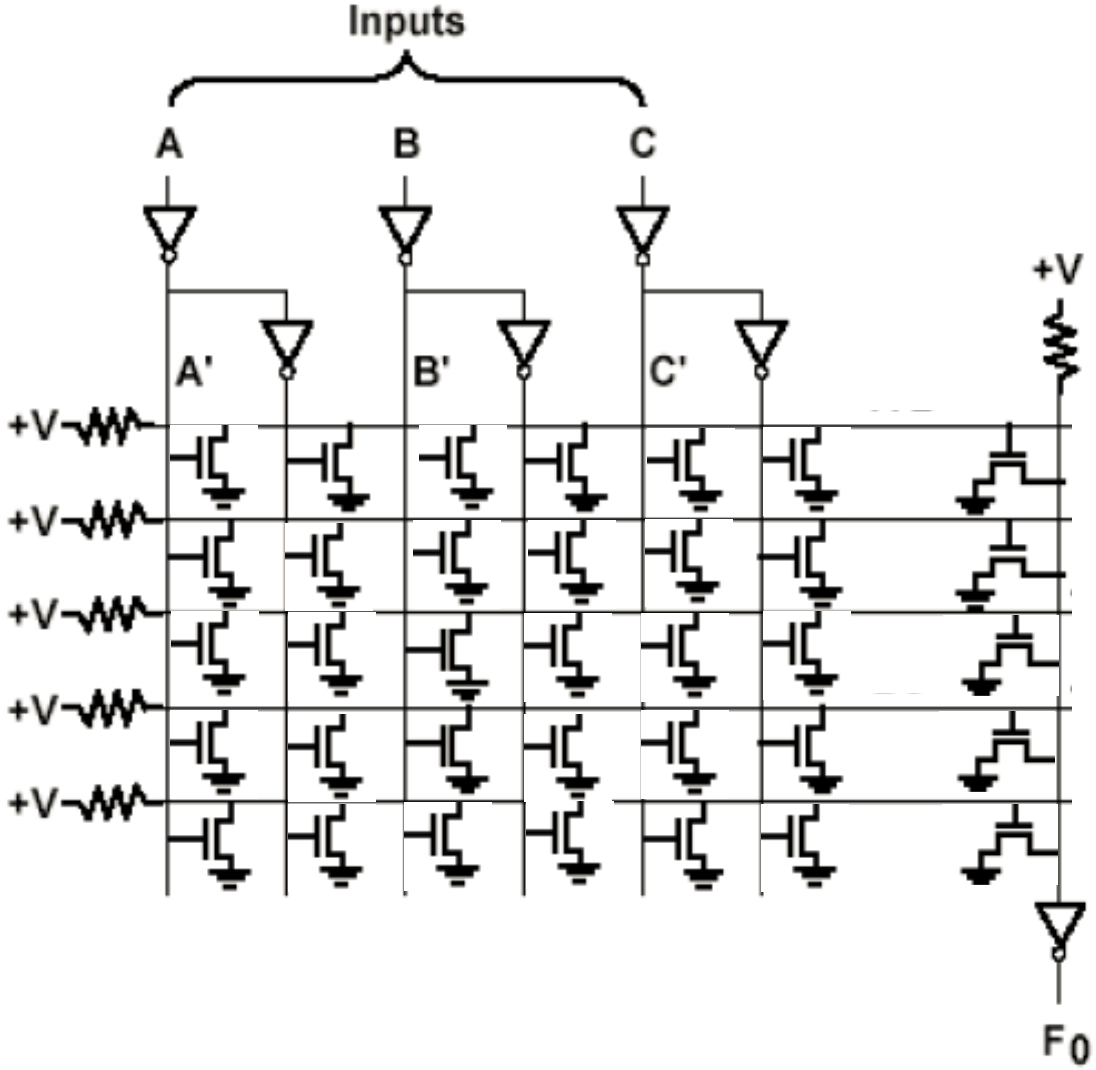
What is required ?

- Process Design Kit (PDK): the set of layers and rules to draw a layout
  - PDKs usually prepared by silicon foundries;
  - Access requires signing Non-Disclosure-Agreements (NDAs);
  - Some free PDKs exist, e.g. SkyWater 130.
- A tool to design and simulate circuits/gates exploiting a PDK.
  - Tools used at industrial level licensed paying fees.
  - Some free tools exist, e.g. Magic and kLayout.

VLSI design methodologies

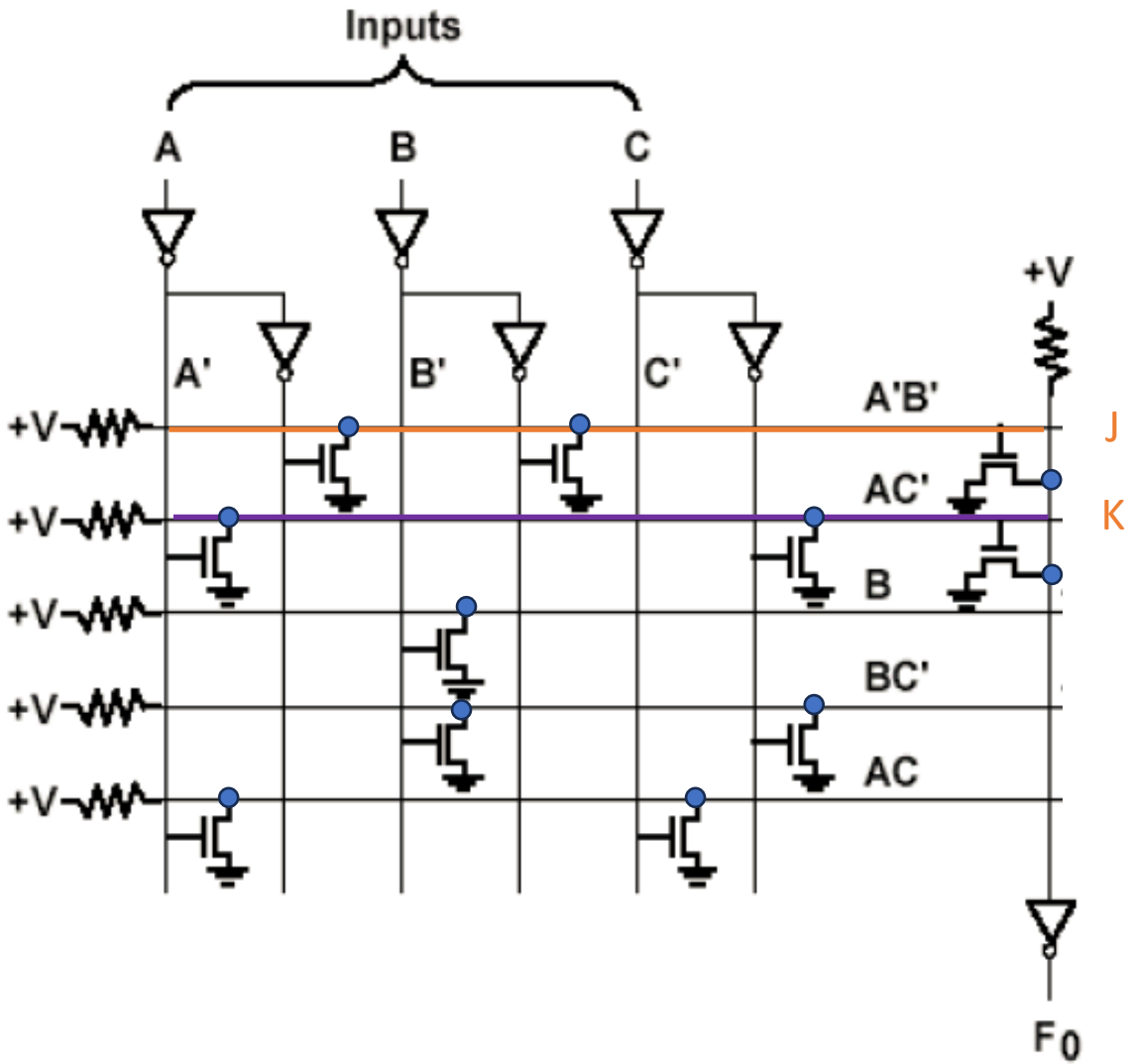
- The methodology we saw so far:
  - Contains the basis steps to fully design and optimize your circuit from scratch
  - Is typical of Application-Specific Integrated Circuits (ASICs)
- However, with few transistors one can build NAND and NOR gates
  - Is its enough to build any combinational logic function !
  - Instead of designing all the transistors from scratch rely on pre-built arrays.
  - Programming the connection obtain arbitrary functions
  - Typical approach to build programmable devices (PAL, PLA, CPLD, FPGA)

Example: programmable device - I



$$F_0 = \overline{A} \cdot \overline{B} + A \cdot \overline{C}$$

Example: programmable device - II



$$J = \overline{A + B} = \overline{A} \cdot \overline{B}$$

$$K = \overline{\overline{A} + C} = A \cdot \overline{C}$$

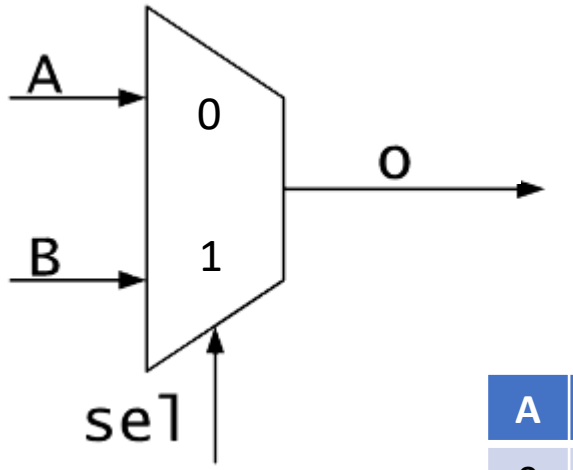
$$F_0 = J + K = \overline{A} \cdot \overline{B} + A \cdot \overline{C}$$

Digital electronics

- Combinational circuits
  - Circuits with no memory: $O_t=f(I_t)$
  - Examples: gates (NOT, NAND, AND, …), selectors (multiplexers), arithmetic operations, …
- Sequential circuits
  - Circuits with memory behaviour: $O_t=g(I_t,O_{t-k})$
    - Asynchronous: the output can change in any time slot, without any reference
    - Synchronous: output changes are synchronized by a reference signal, usually referred to as clock (square wave with duty cycle 50%)
      - Level triggered: latch-like behaviour (50% transparence phase - 50% memory phase)
      - Edge triggered: flip-flop-like behaviour (istantaneous transparence phase – long memory phase)
        - Positive edge triggered (rising edge)
        - Negative edge triggered (falling edge)
        - Double Data Rate (both rising and falling)

Combinational circuits example

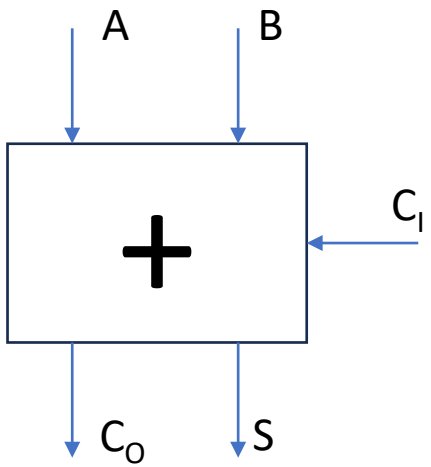
- Multiplexer



A	B	sel	O
-	-	0	A
-	-	1	B

$$O = A \cdot \overline{sel} + B \cdot sel$$

- Full adder



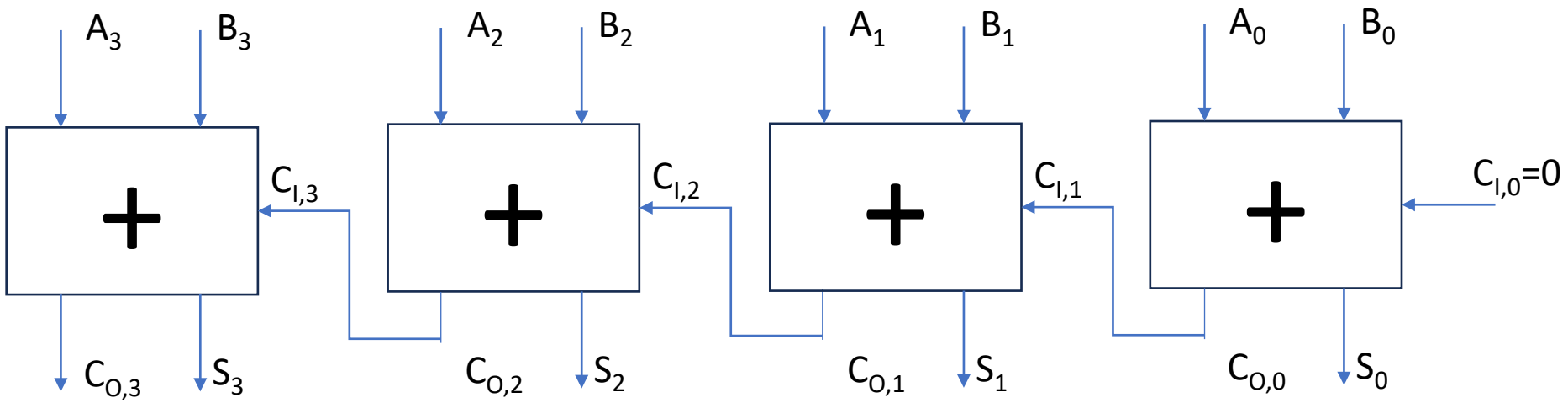
A	B	C_I	S	C_O
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = A \oplus B \oplus C_i$$

$$C_O = A \cdot B + A \cdot C_I + B \cdot C_I$$

More on combinational circuits

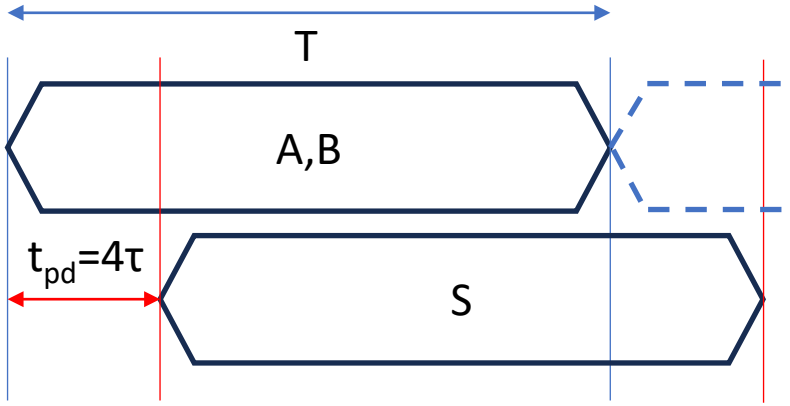
- Important parameter: combinational delay
  - The output is not updated istantaneusly
  - There is a delay between changes in the input (triggering a change in the output) and actual output update
- Example:
  - 4bit adder



Combinational delay

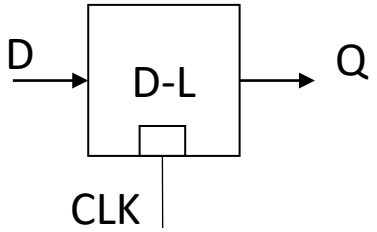
- Assume one full adder introduces a propagation delay τ
  - S_3 delay is 4τ
  - Cascading combinational blocks increases the propagation delay (t_{pd})

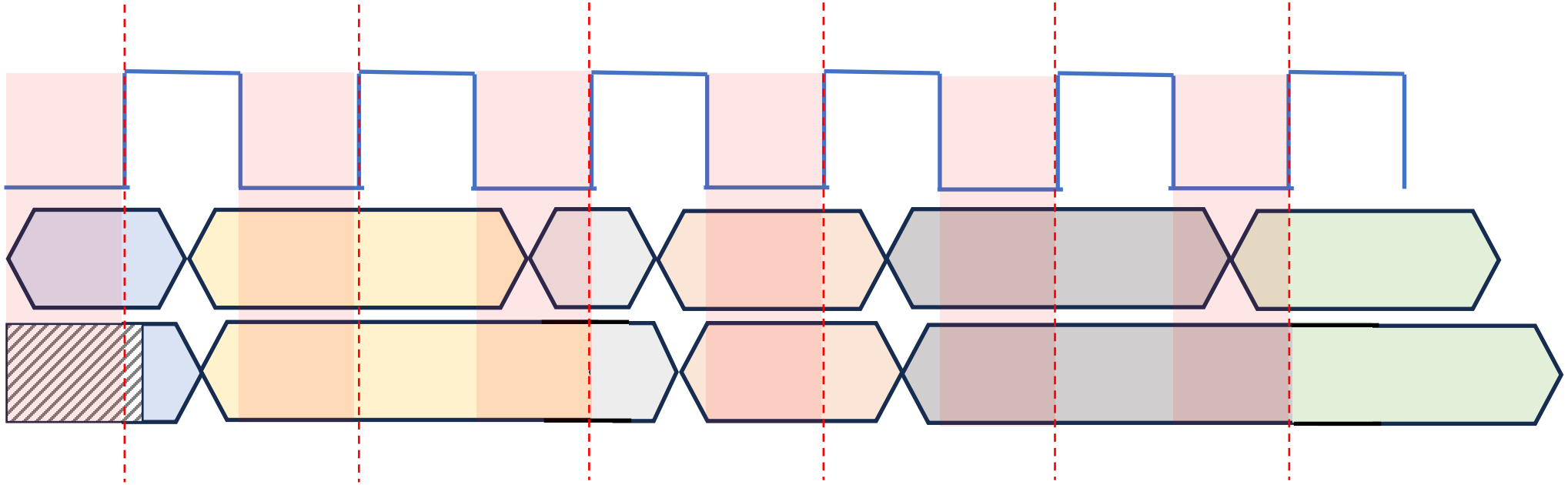
- If data change at a certain rate T, the maximum combinational delay must guarantee correct results: $t_{pd}<T$



Sequential circuits - I

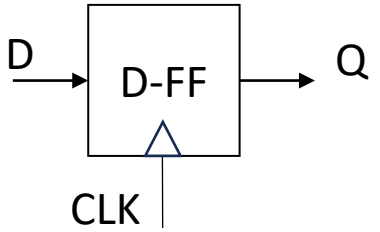
- Level triggered (Latch-like)
  - Simplest example: D-type latch
    - When the clock is low → Memory
    - When the clock is high → Transparency

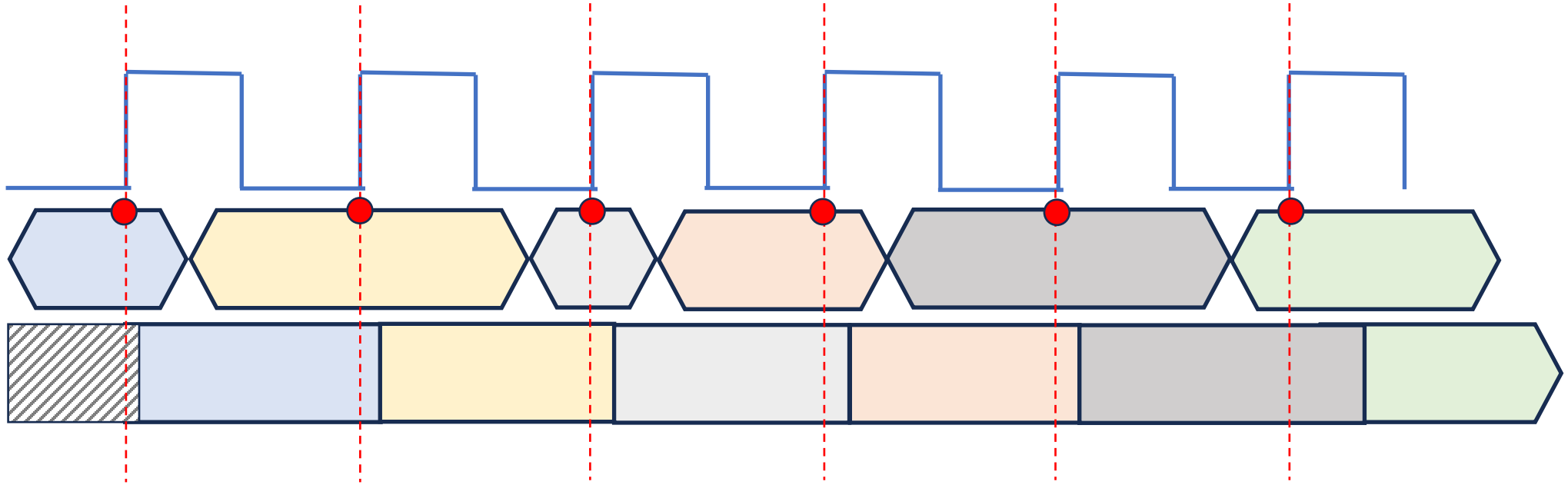




Sequential circuits - II

- Positive edge triggered (Flip-Flop-like)
  - Simplest example: D-type Flip-Flop
    - Always in Memory
    - Except on the rising edge of the clock → Transparency

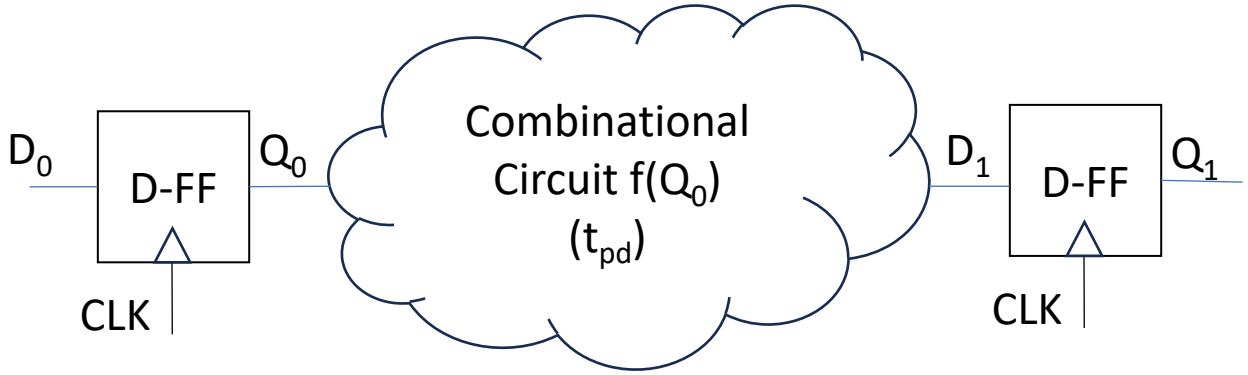


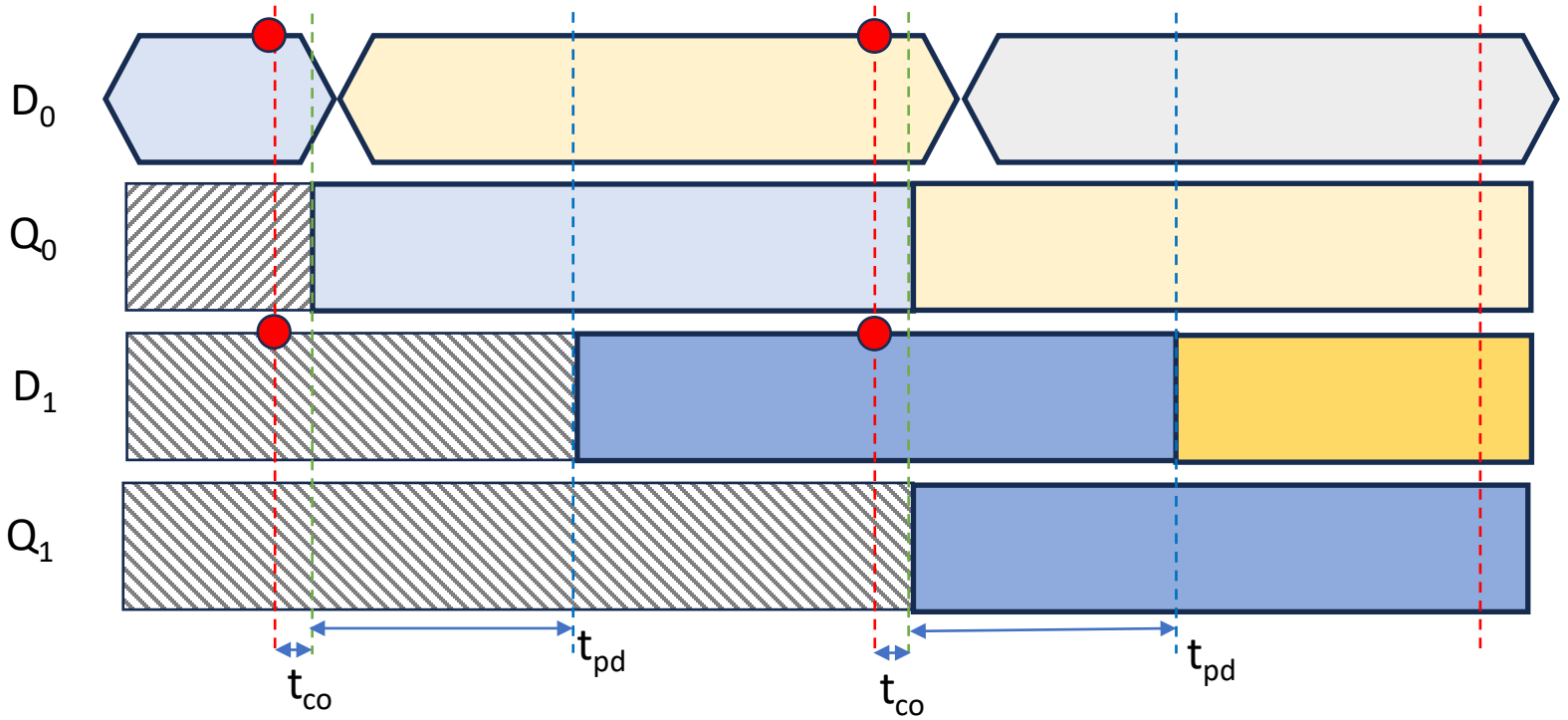


Sequential circuits - II

- Level triggered Vs positive edge triggered
  - Level triggered elements have a direct path between input and output during transparency phase
  - Cascade of level triggered elements in the transparency phase behaves as a chain of combinational circuits (potentially long t_{pd})
    - It can be avoided cascading sequences of positive/negative level triggered elements
      - Complex especially if there is a feedback
  - Positive edge triggered elements can be cascaded (no effect on the delay)
    - Increasing latency
    - Easy calculation of the maximum speed of the circuit.

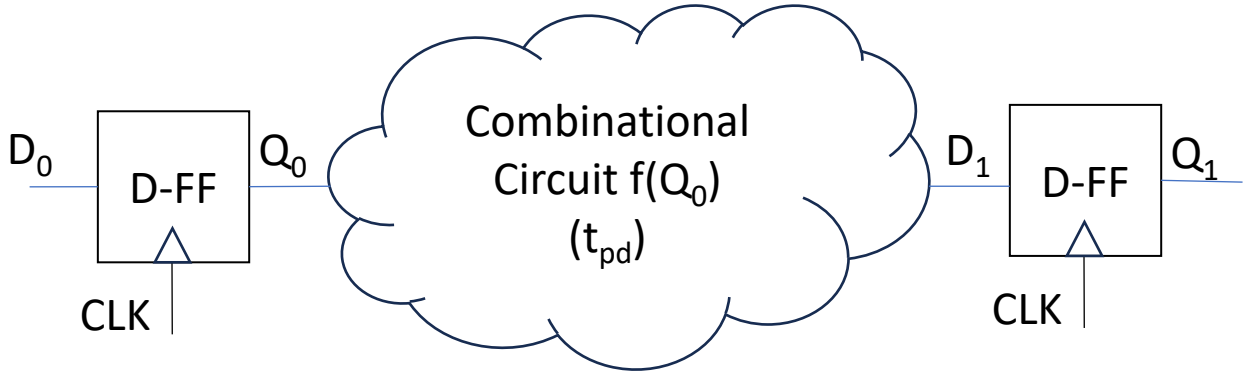
Example: circuit with FFs - I

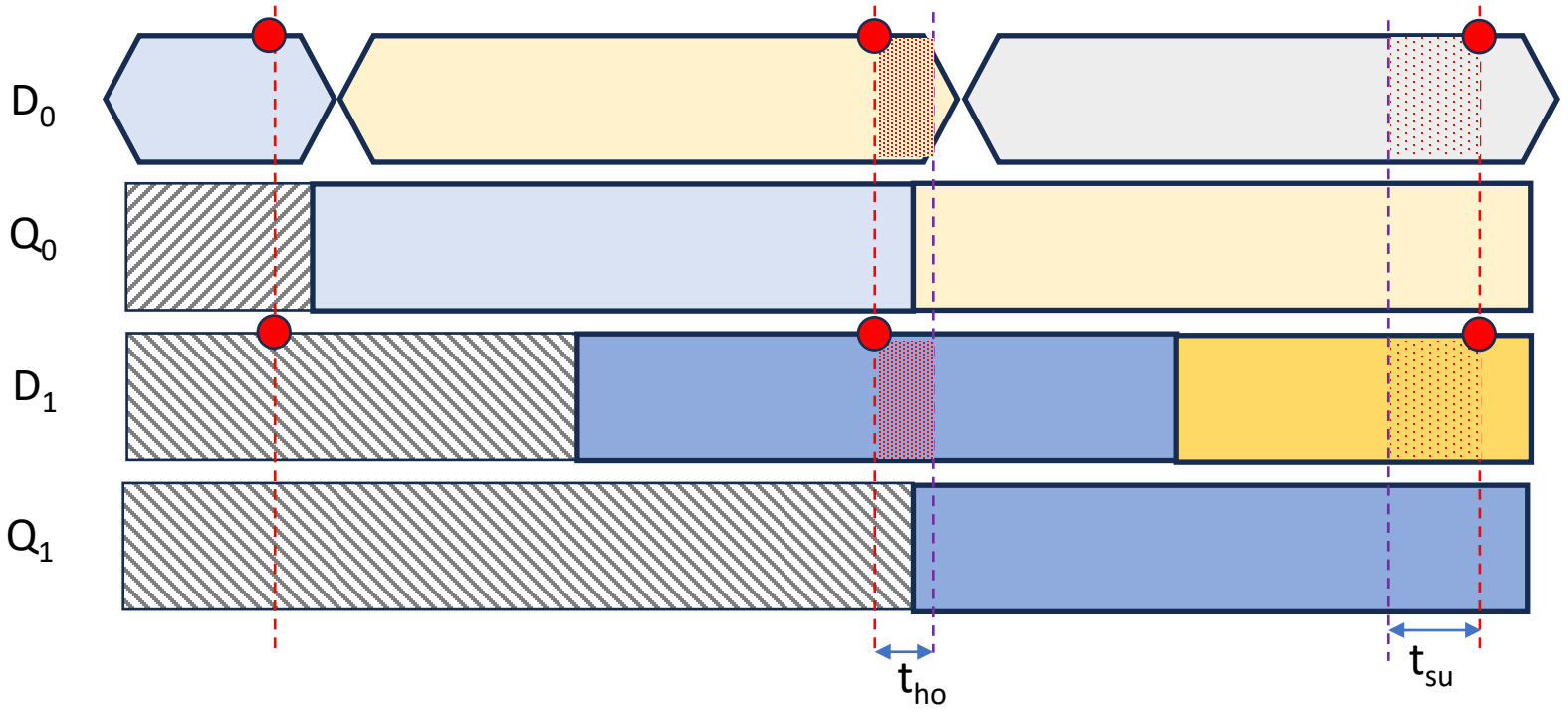




- t_{co} → Clock-to-output delay

Example: circuit with FFs - II



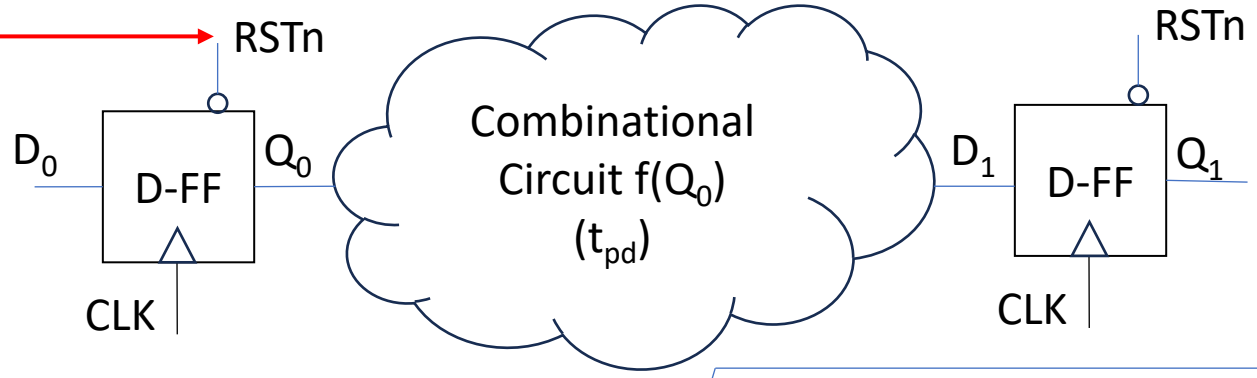


- t_{co} → Clock-to-output delay
- t_{su} → Setup time
- t_{ho} → Hold time

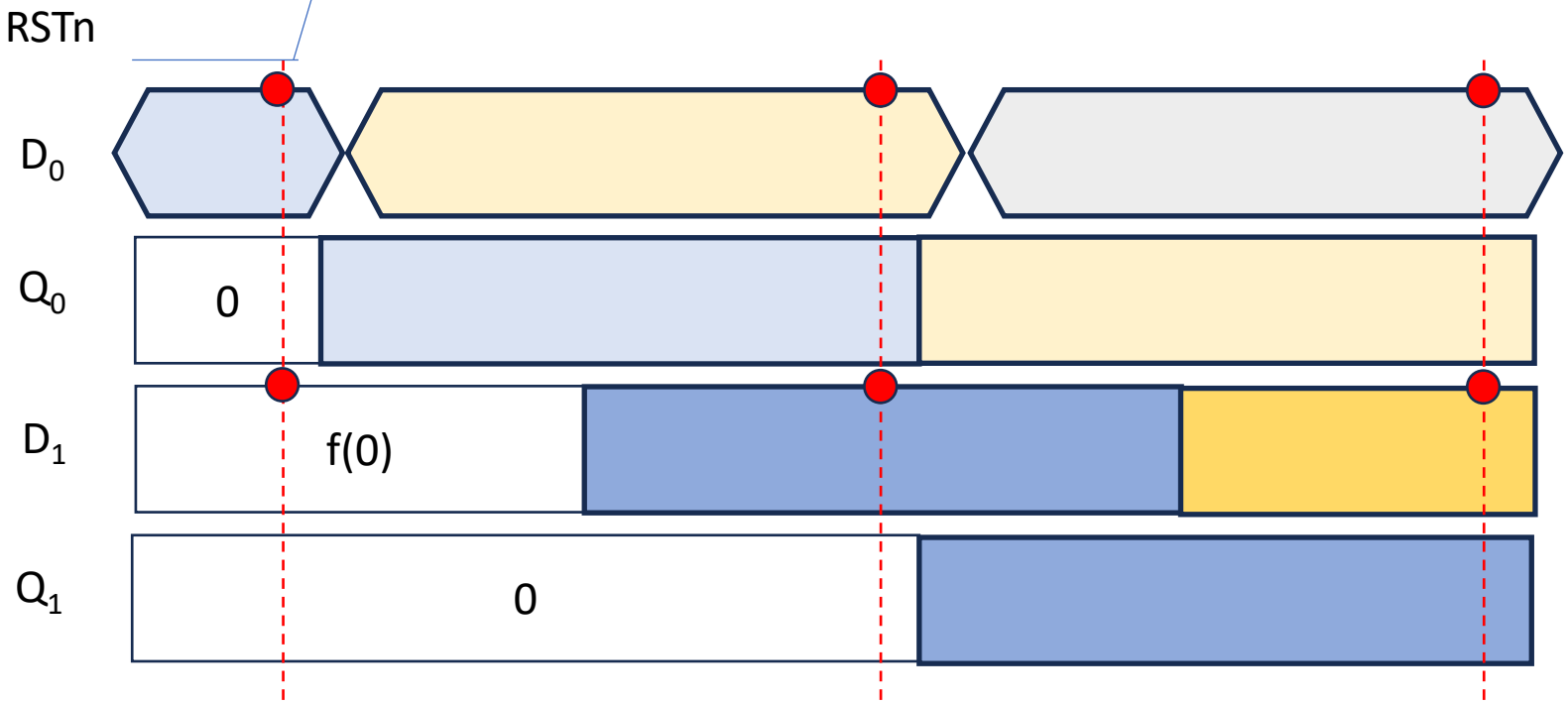
$T_{clk} \geq t_{co}+t_{pd}+t_{su}$

$t_{co}+t_{pd} \geq t_{ho}$

Example: circuit with FFs - III

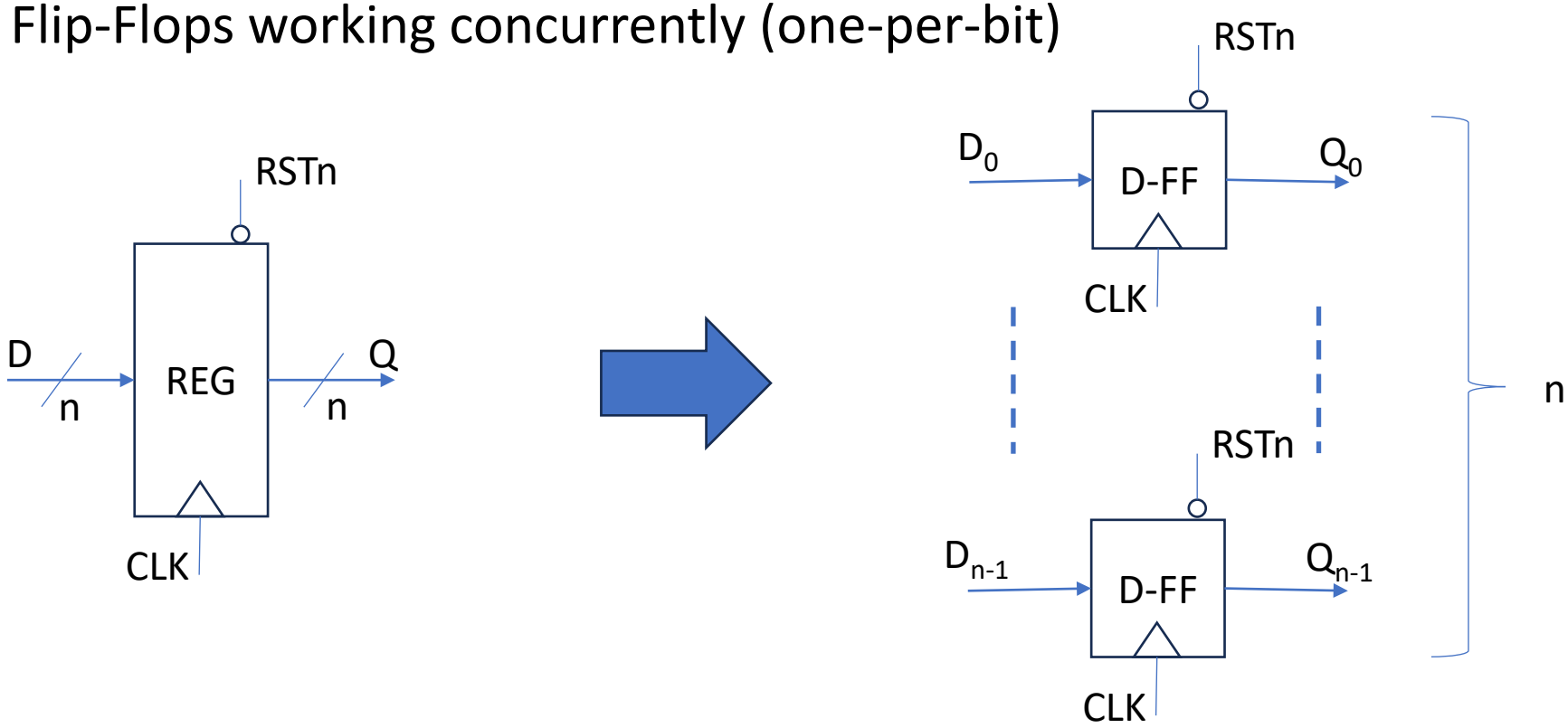


- Asynchronous reset
- Usually active-low



From Flip-Flops to registers

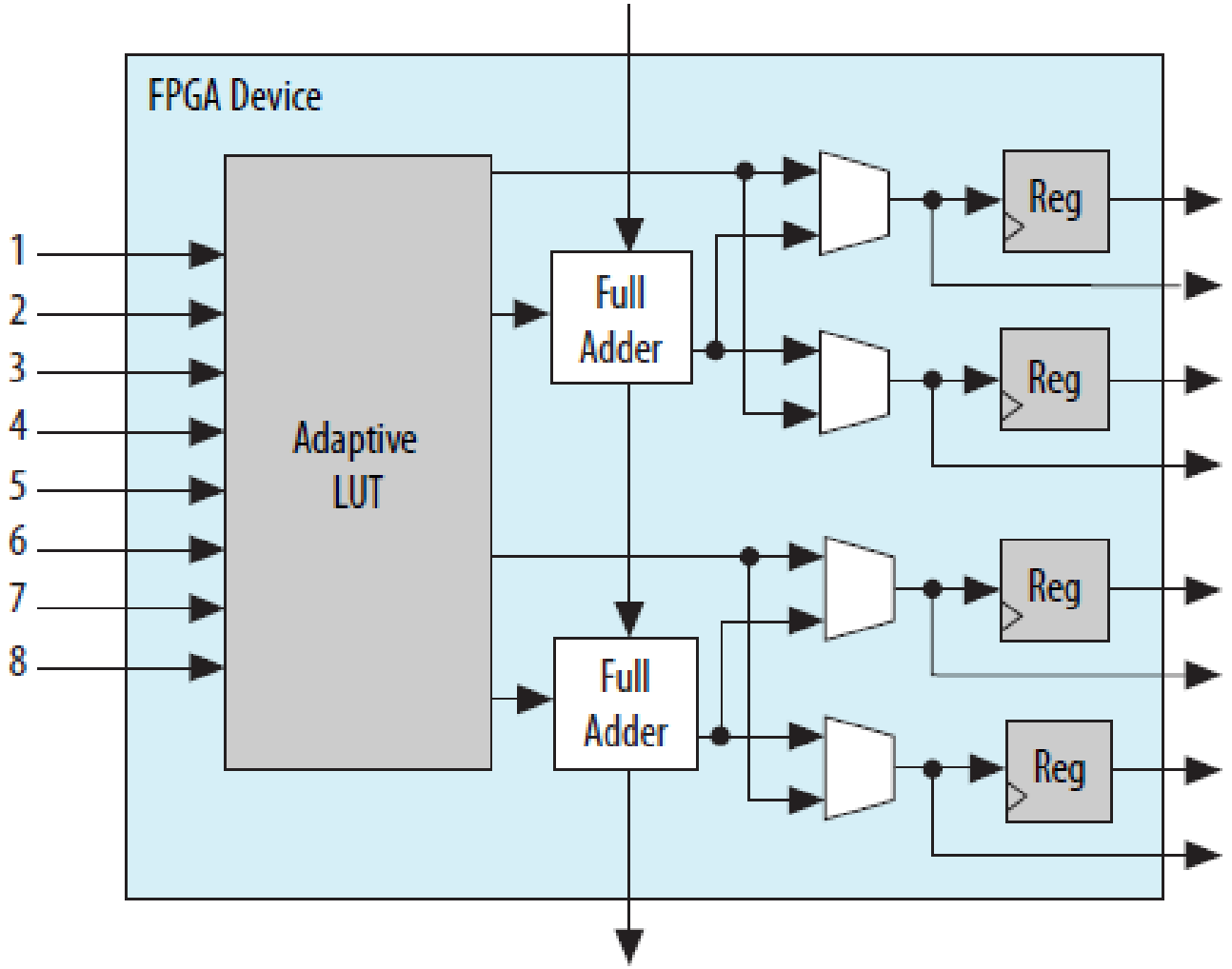
- Flip-Flop → 1-bit-data
- Register (parallel register) → n-bit-data
  - n Flip-Flops working concurrently (one-per-bit)



VLSI design methodologies (continue)

- The methodology we saw for designing transistors (W/L) can still be applied … but the number of transistor has increased.
- Adding sequential circuits to a programmable combinational structure increases the potential of the programmable device.
  - Modern programmable devices (FPGAs) replaced the NAND/NOR programmable combinational structure with Look-Up Tables (LUTs)

Example: basic cell of an Intel-Altera FPGA

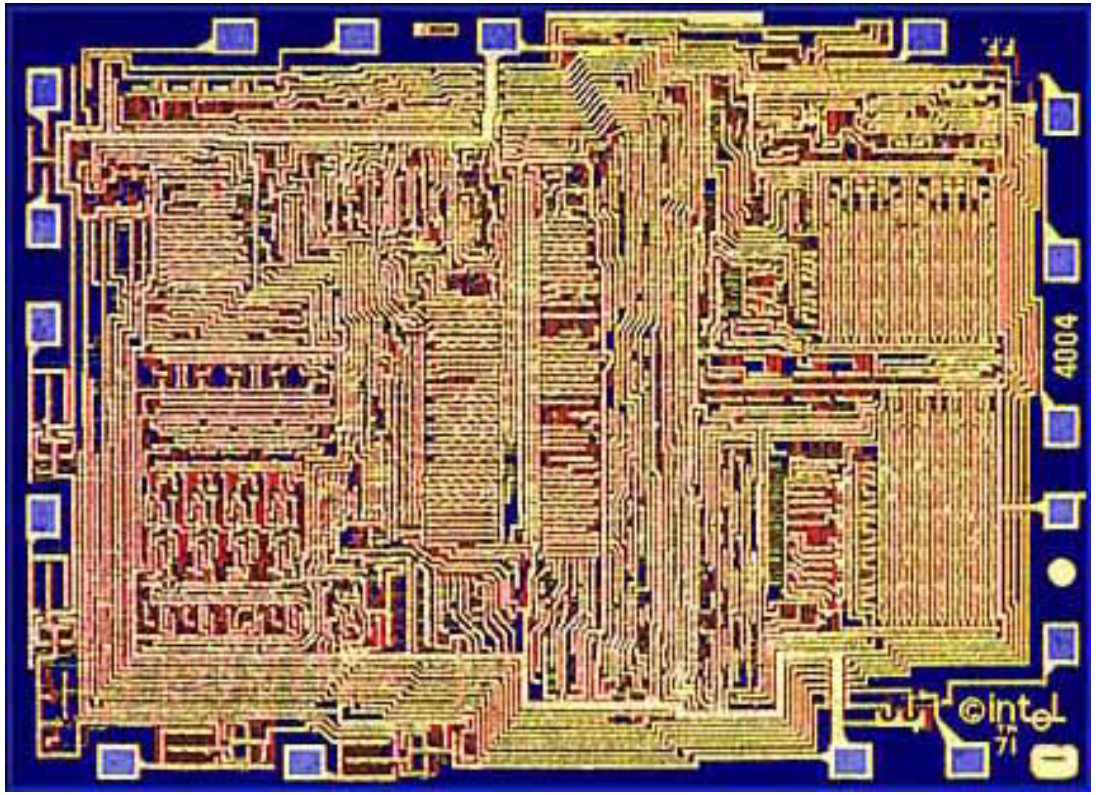


VLSI design styles

- Custom Design
- Semi-Custom Design
- Programmable Logic Devices (PLDs)

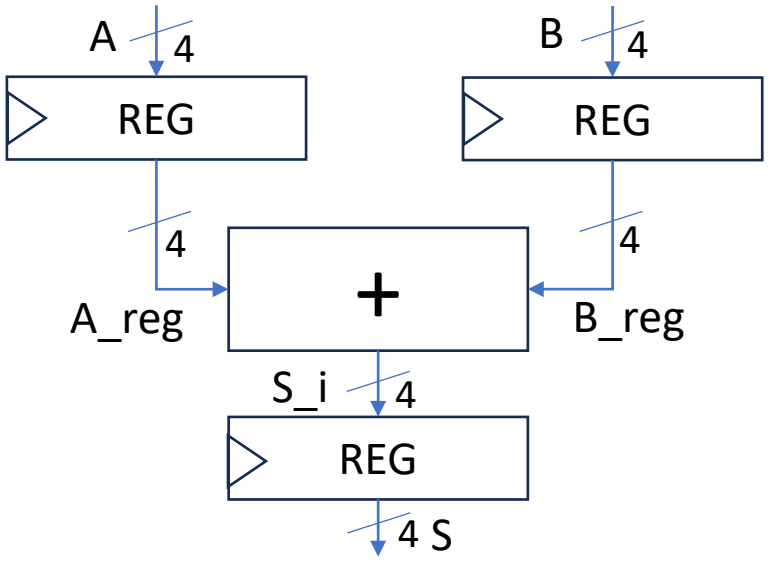
Custom Design

- Extension of the methodology seen so far (W/L).
- Viable for small designs or critical parts.
- Example: INTEL 4004 (1971)

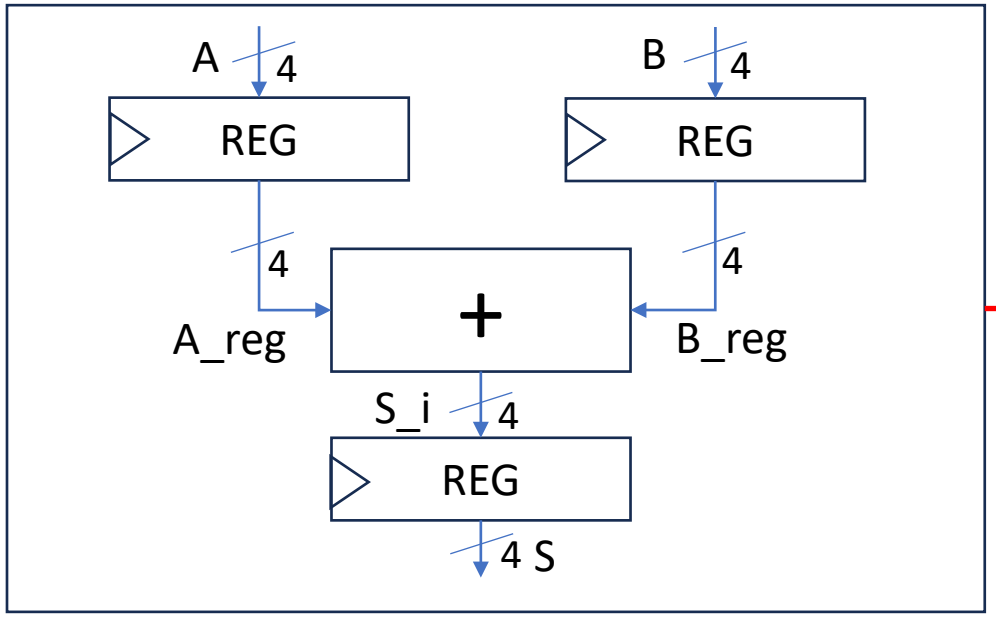


Semicustom design

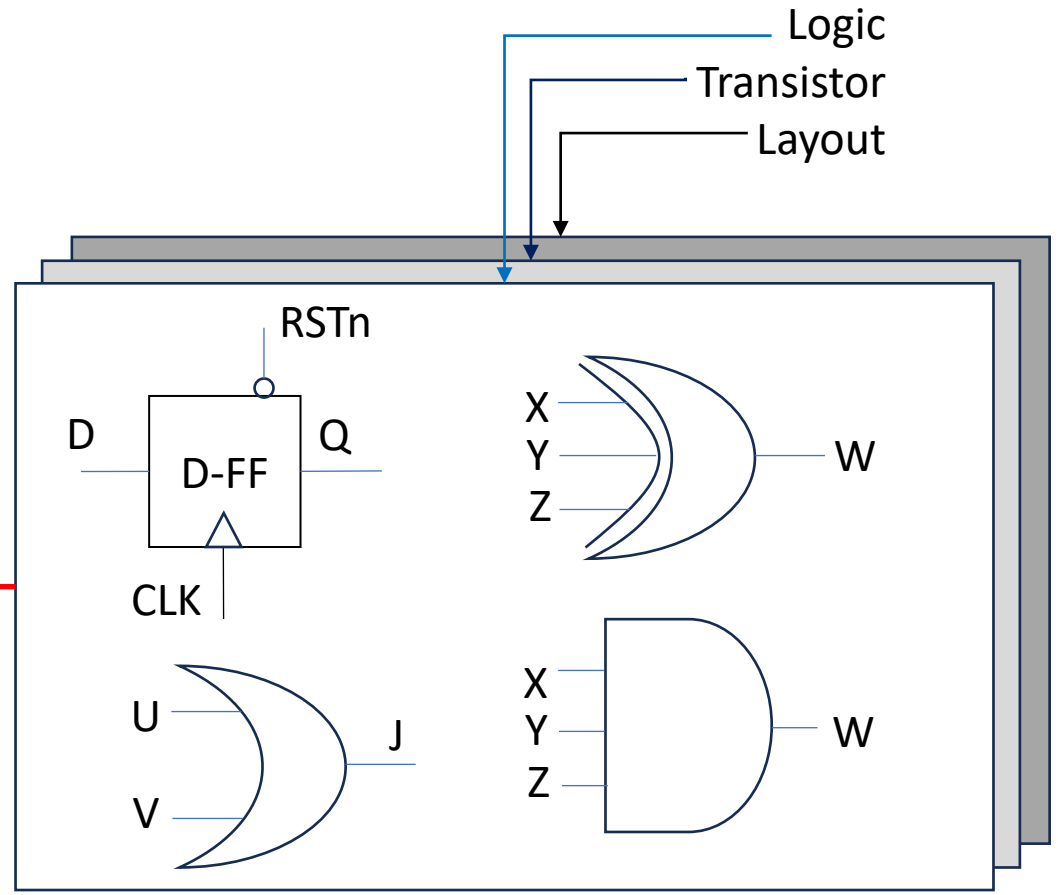
- For mid/complex designs:
  - Rely on a library of cells already made by someone else (silicon foundry)
  - Use them as bricks to build the circuit.
  - Example:
    - 4 bit adder with input and output registers



Example - I



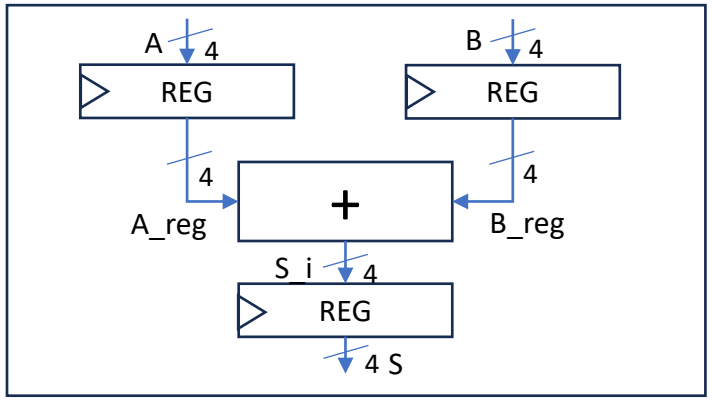
Register Transfer Level (RTL) description

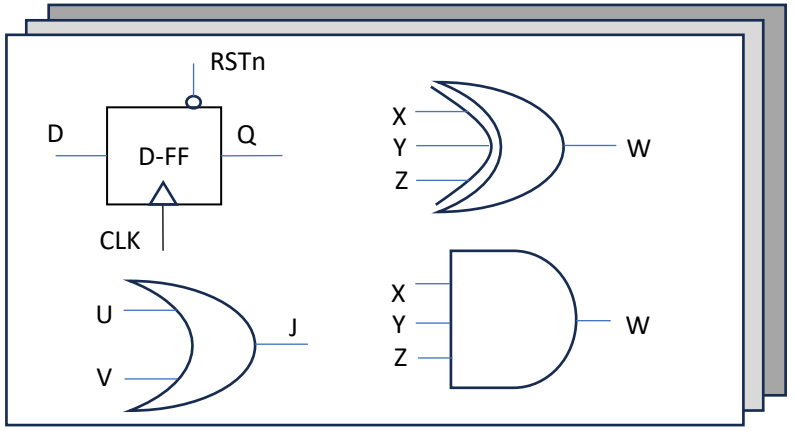


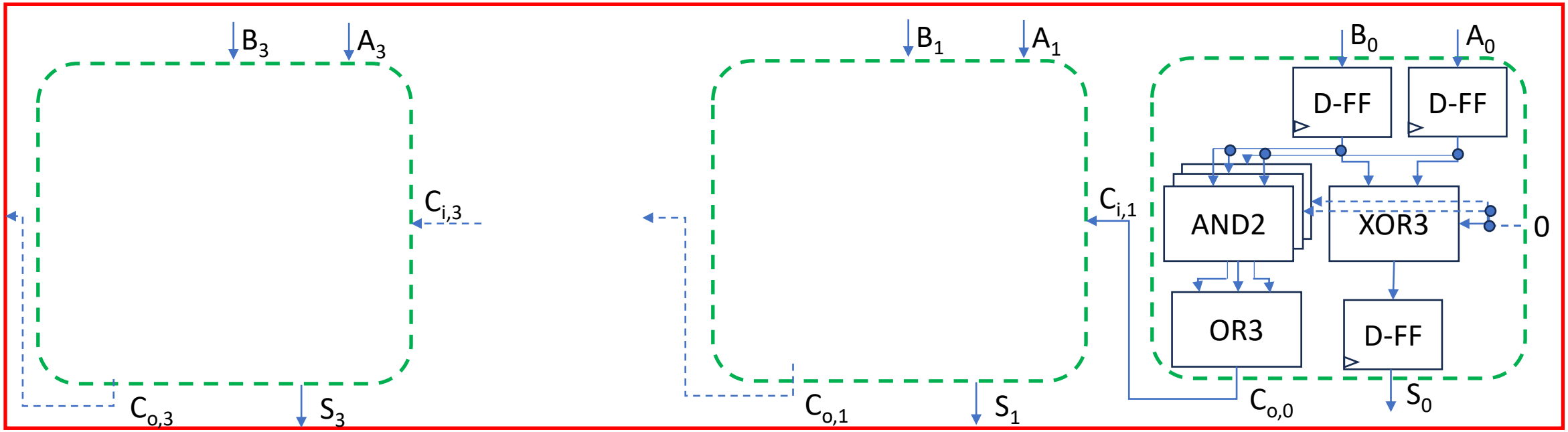
Library of cells

Netlist

Example - II

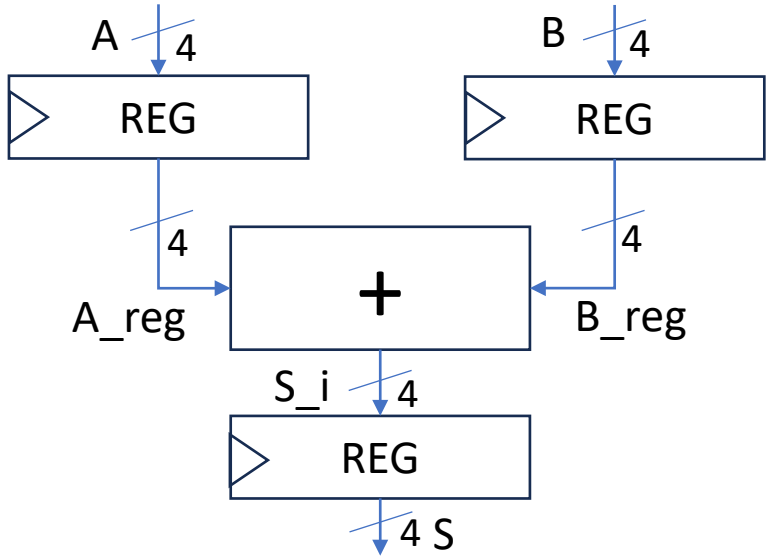

A
4
B
4
REG
REG
4
4
A_reg
+
B_reg
S_i
4
REG
4 S


RSTn
D
D-FF
Q
CLK
X
Y
Z
W
U
V
J
X
Y
Z
W

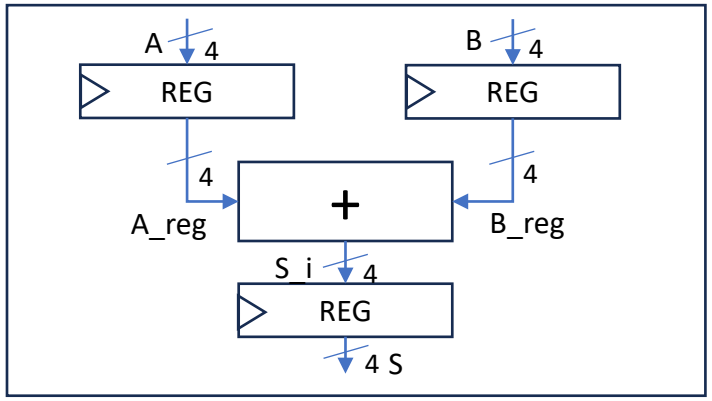

B3
A3
Ci,3
Co,3
S3
B1
A1
Ci,1
Co,1
S1
B0
A0
D-FF
D-FF
AND2
XOR3
0
OR3
D-FF
Co,0
S0

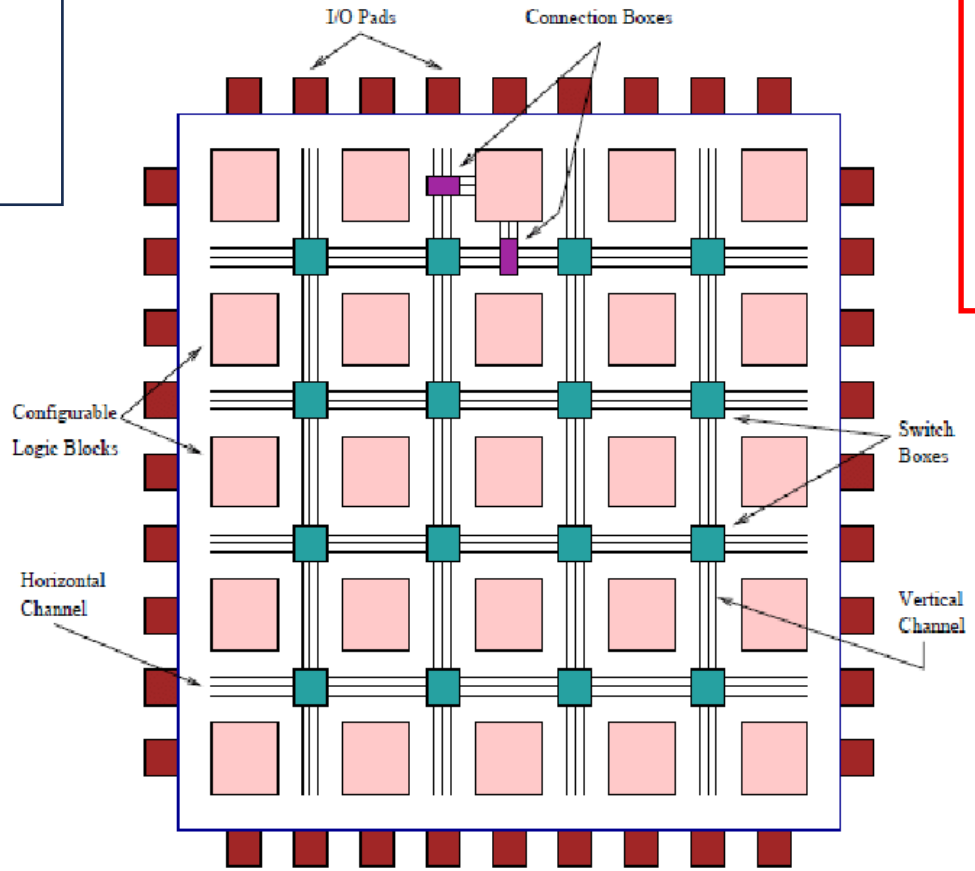
Programmable logic devices

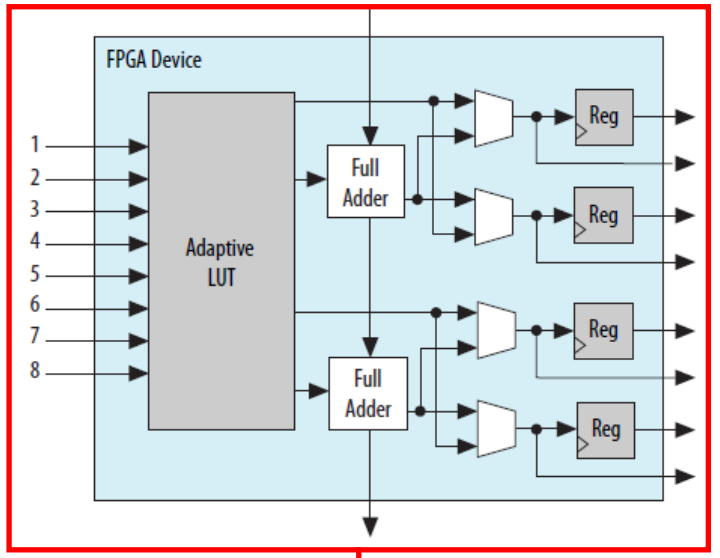
- For simple to complex designs:
  - Rely on an array of programmable cells already made by someone else (PLD vendor)
  - Use them as bricks to build the circuit.
  - Example:
    - 4 bit adder with input and output registers



Example - I



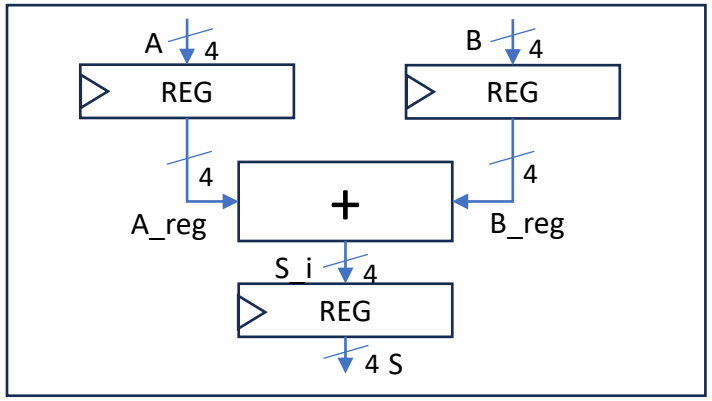


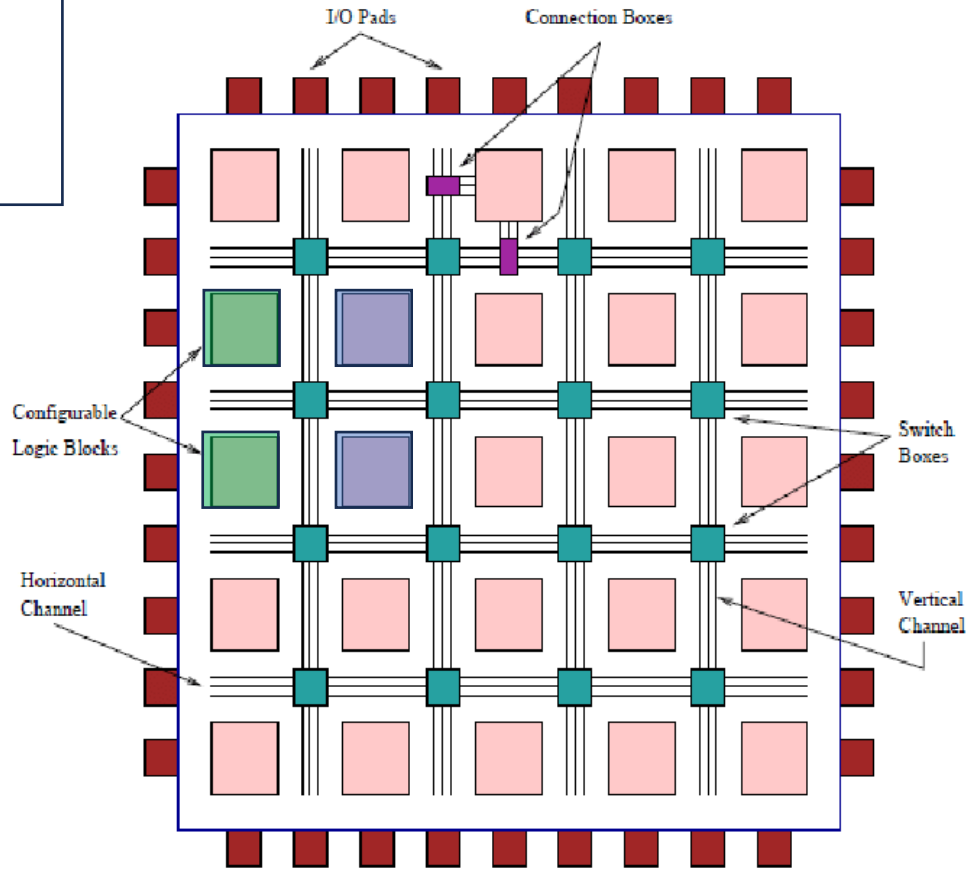


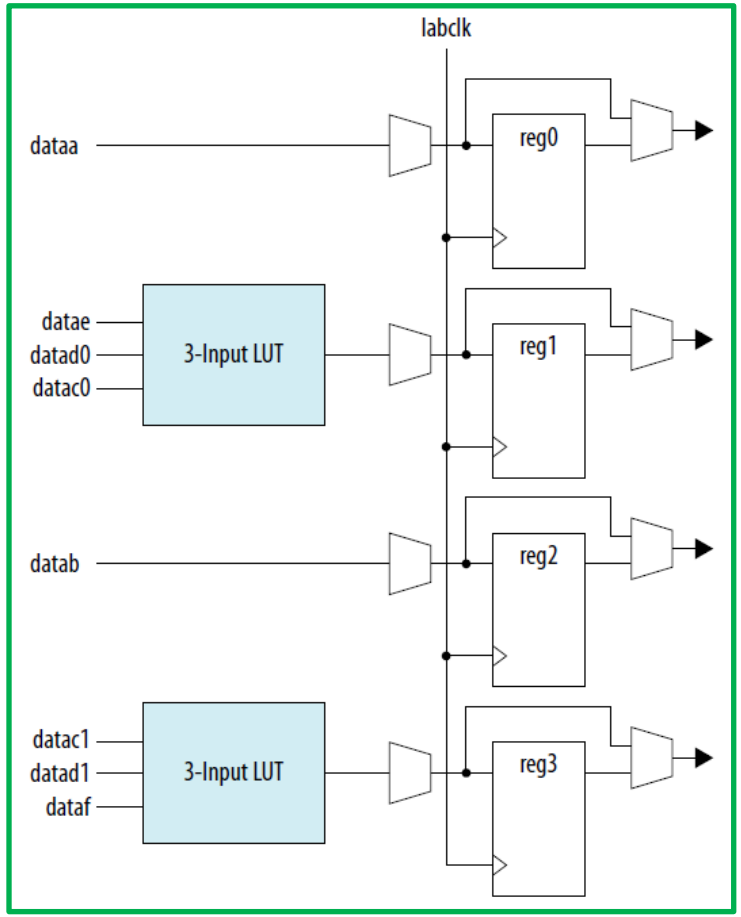
Can be configured in different modes

Taken from «Design FPGA-Based CL-Minimum Control Unit» by F. Piltan et al.

Example - II

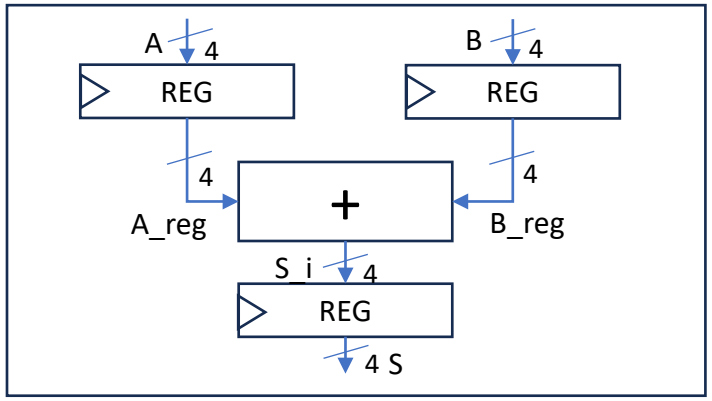


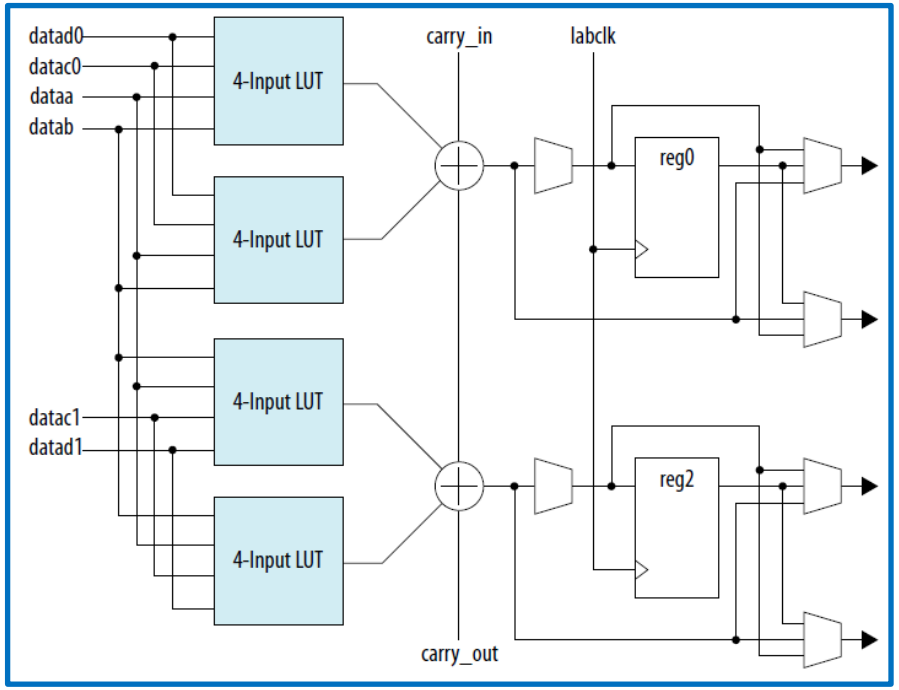


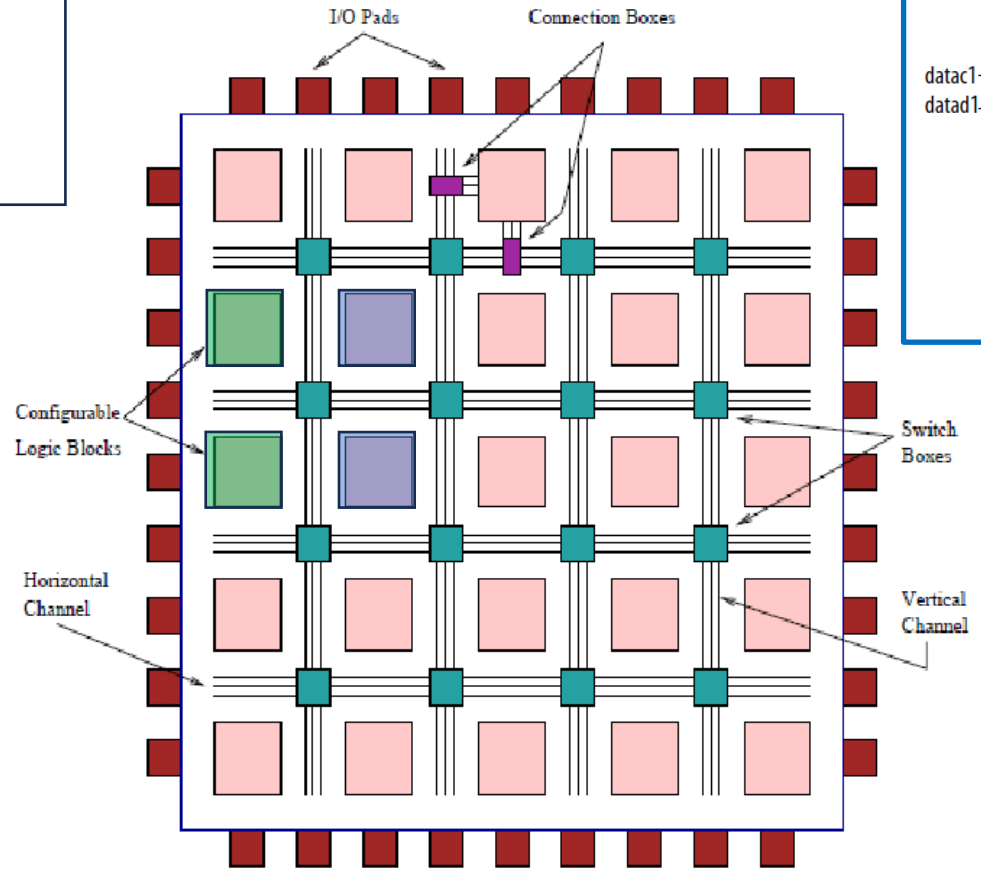


Taken from «Design FPGA-Based CL-Minimum Control Unit» by F. Piltan et al.

Example - III







Taken from «Design FPGA-Based CL-Minimum Control Unit» by F. Piltan et al.

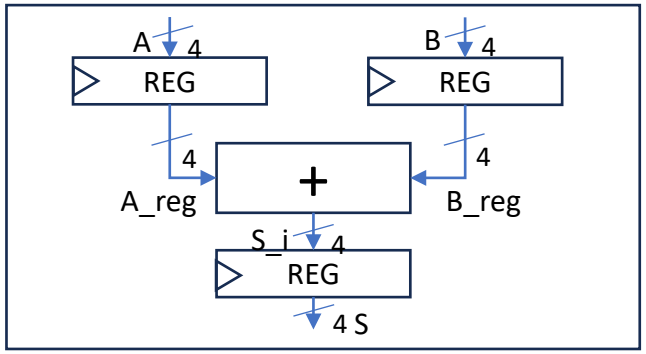
Design flow

- If the circuit complexity increases, manually translate RTL to logic becomes impossible !
- Exploit Computer Aided Design (CAD) instruments: logic synthesizers
- Elements:
  - RTL description
  - Technology information (standard cells or PLD cells)
  - Constraints (area, timing, power)

RTL description

- Circuit description via a Hardware Description Language (HDL)
  - VHDL
  - Verilog/SystemVerilog
- Inspired by object oriented programming languages, but designed for hardware description.
- Clear separation between interface and implementation

Interface



- Name the element and list signals, directions and types

```vhdl
entity add4 is
  port (
    A   : in  std_logic_vector(3 downto 0);
    B   : in  std_logic_vector(3 downto 0);
    CLK : in  std_logic;
    S   : out std_logic_vector(3 downto 0));
end entity add4;
```

VHDL

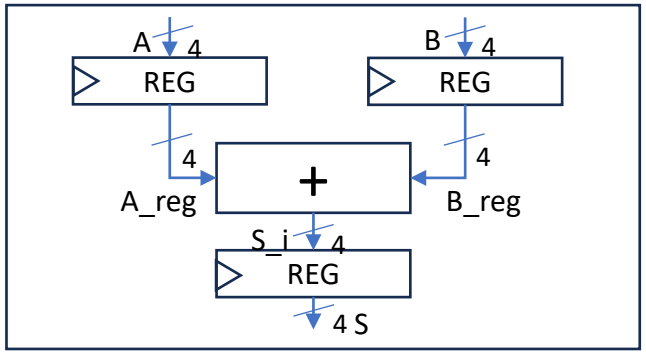
```verilog
module add4 (input [3:0] A,
             input [3:0] B,
             input CLK,
             output reg [3:0] S);

/* implementation */

endmodule
```

Verilog

Implementation

- Contains only three possible elements (acting concurrently):
  - Components
    - Hierarchical design
  - Processes
    - To model sequential elements
    - To model (mainly) combinational decoders
  - Assignments
    - To model simple combinational logic (mainly arithmetic and logic functions)

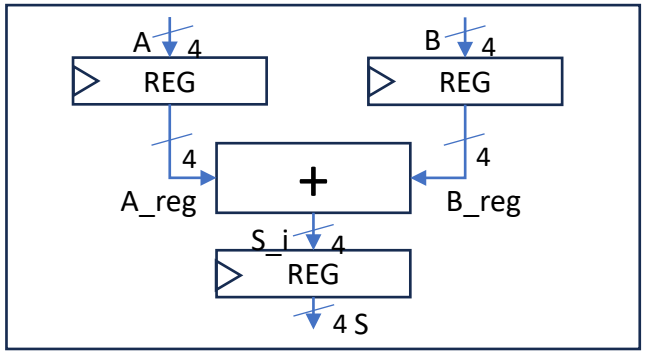
VHDL



- Use of process and assignment

```
architecture beh of add4 is

  signal A_reg : std_logic_vector(3 downto 0);
  signal B_reg : std_logic_vector(3 downto 0);
  signal S_i : std_logic_vector(3 downto 0);

begin  -- architecture beh

  process (CLK) is
  begin  -- process
    if CLK'event and CLK = '1' then  -- rising clock edge
      A_reg <= A;
      B_reg <= B;
      S <= S_i;
    end if;
  end process;

  S_i <= A_reg + B_reg;

end architecture beh;
```

Verilog



- Use of process and assignment

```
always @ (posedge CLK) begin
     A_reg <= A;
     B_reg <= B;
     S <= S_i;
  end

  assign S_i = A_reg + B_reg;
```

Design flow (continue)

- Elements:
  - ✓RTL description
  - ✓Technology information (standard cells or PLD cells)
    - Standard cells
      - usually prepared by silicon foundries;
      - Access requires signing Non-Disclosure-Agreements (NDAs);
      - Some free sets of standard cells exist, e.g. SkyWater 130.
    - PLD
      - vendors provide just enough details to developers
      - Simpler/older devices often for free
      - Complex/modern devices require fees
  - ✓Constraints (area, timing, power)
    - Imposed by the designer based on the application requirements
    - Example, Synopsys Design Constraints (sdc)

```
create_clock -name MYCLK -period 3.5 CLK
```

Logic synthesizers

- Tools used at industrial level licensed paying fees.
- ASIC
  - Some free tools exist, e.g. YOSYS
- PLD
  - Development tools are usually offered for free with limited features
  - E.g.
    - Vivado/Vitis by AMD/Xilinx
    - Quartus Prime by Intel/Altera
    - …

Logic synthesis - I

- Provide the tool with area, timing, power information about the cells.
- Example, liberty file:

```
library ("library_name") {
    …
    /* Units */
    time_unit : "1ns";
    voltage_unit : "1V";
    leakage_power_unit : "1nW";
    current_unit : "1mA";
    …
    cell ("nand2") {
       area : 3.7536000000;
       …
       /* timing and power data */
       …
```

Logic synthesis - II

1. Read area/timing/power cells file
2. Read the HDL description and map it to a generic RTL description.
3. Read constraints.
4. Map/optimize the RTL description on the available cells
5. Write the resulting netlist

Example

```
module add4 (A, B, CLK, S);
        input [3:0] A;
        input [3:0] B;
        input CLK;
        output reg [3:0] S;
        wire n0;
        wire n1;
        wire n2;

 dff i0 (
    .CLK(CLK),
    .D(A[0]),
    .Q(n0));

dff i1 (
    .CLK(CLK),
    .D(B[0]),
    .Q(n1));
```

```
xor2 i2 (
    .A(n0),
    .B(n1),
    .X(n2));

dff i3 (
    .CLK(CLK),
    .D(n2),
    .Q(S[0]));

…

endmodule
```

Netlist to layout ?

- The netlist contains the list of required cells and how they are connected together.
- Given the constraints and and the geometrical details of the cells (Library Exchange Format - LEF file), one can place and connect (route) them.
- Tools for Place & Route: the ones used at industrial level licensed paying fees.
  - ASIC
    - Some free tools exist, e.g. OPENROAD
  - PLD
    - Development tools are usually offered for free with limited features
      - Vivado/Vitis by AMD/Xilinx
      - Quartus Prime by Intel/Altera
      - …

ASIC P&R

- Complex part of the design flow made of three main steps.

1. Floorplanning
2. Placement
3. Routing

Floorplanning

- Define chip organization:
  - Pin position
  - Power supply
  - Area for cells
  - Area for routing
  - …

Placement

- Know area reserved for cells placement → from flooplanning
- Know layers and cells geometry → lef files

```
/* layers definition */
LAYER nwell
        …
END nwell
…
```

```
/* cells geometry definition */
MACRO nand2
  CLASS CORE ;
  ORIGIN 0.000 0.000 ;
  SIZE 1.380 BY 2.720 ;
  …
  PIN A
    DIRECTION INPUT ;
    …
    PORT
      LAYER li1 ;
        RECT 0.940 1.075 1.275 1.325 ;
    END
  END A
  …
END nand2
```

Note

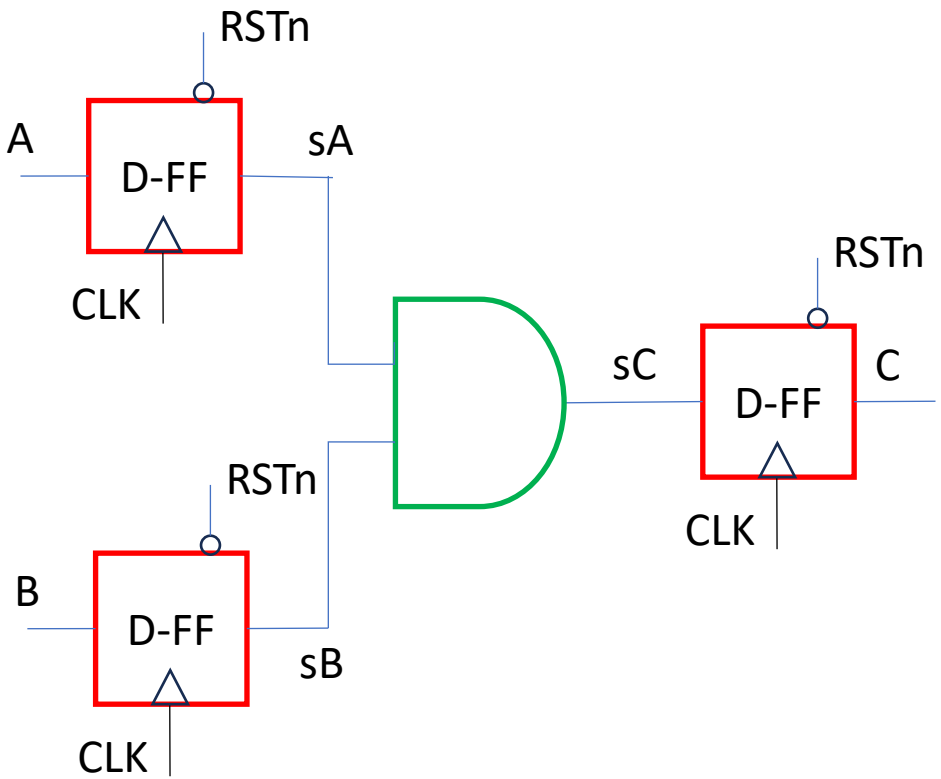
- In order to accommodate different cells in one row, cells have the same height and different width:
- E.g. (from LEF file)
  - NAND2 → SIZE 1.380 BY 2.720
  - D-FF → SIZE 9.200 BY 2.720

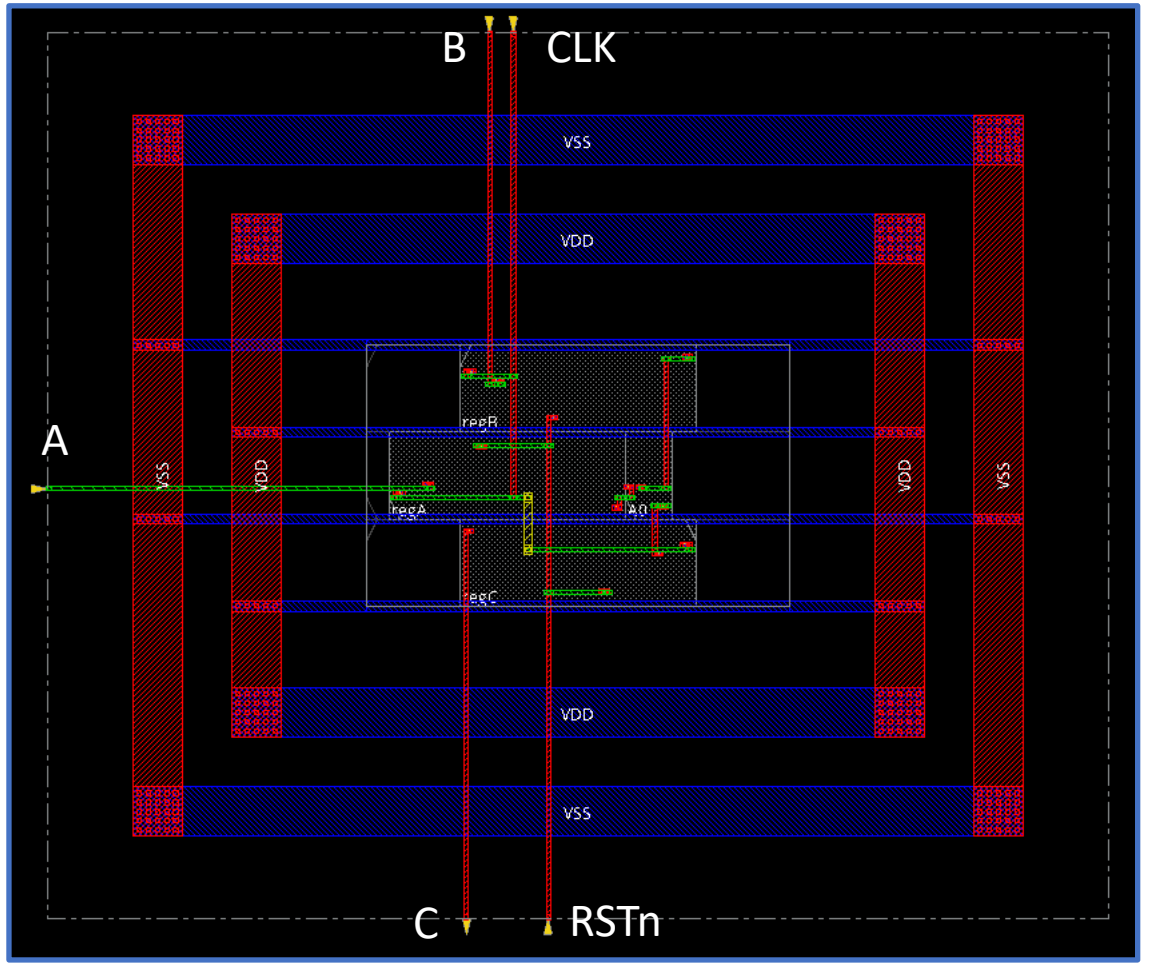
Routing

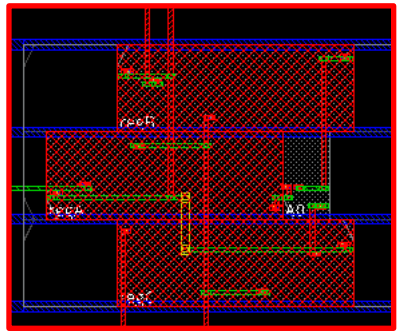
- Know area reserved for wires → from flooplanning
- Know metal layers geometry → lef files

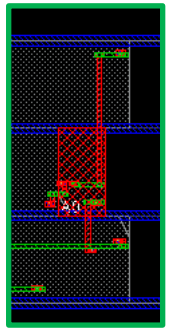
```
LAYER met1
  TYPE ROUTING ;
  DIRECTION HORIZONTAL ;
  …
END met1
```

Example


RSTn
A
D-FF
sA
CLK
RSTn
sC
D-FF
C
CLK
RSTn
B
D-FF
sB
CLK


B
CLK
VSS
VDD
A
VSS
VDD
VDD
VSS
VDD
VSS
C
RSTn





FPGA P&R

- Simplified version of the ASIC P&R
  - Structure already defined
  - Find where to place logic to minimize routing delay
- Final result is a bitstream which configures the PLD according with the result obtained after P&R.
- The bitstream is loaded on the PLD
  - SRAM based
  - Flash based
  - …

Intellectual Properties (IPs)

- Reusable units developed to be used as building blocks in different designs.
- E.g.
  - Microprocessors
  - Memories
  - Accelerators
  - Interfaces/protocols
  - …

Advantages and challenges

- Fast time-to-market, reduced cost
- Verification and compatibility
- Security and licensing