CO102
Computer Hardware Experiment

Lecture 01: Introduction

Liang Yanyan

澳門科技大學
Macau of University of Science and Technology

General References

- Instructor: Dr. Liang Yanyan (梁延研)
  - Email: yyliang@must.edu.mo
  - Tel: 88971997
  - Office: A218

- Grading Information
  - Grade determinates
    - Labs 50%
    - Final Exam 50%

- Please submit your lab reports to Moodle.

General References

- www.vhdl.org
- www.fpga.com.cn
- www.opencore.com
-
- Some Resource in FTP
  - Username: yyliang_stu
  - Password: ^%+q2Hkn
  - Link： ftp://ftp.must.edu.mo/

- Textbook:
  - VHDL Design Representation and Synthesis, Armstrong and Gray, Prentice Hall.
  - Digital Design: An Embedded Systems Approch Using VHDL, Peter J. Ashenden, Morgan Kaufmann.

Outline

Index	Topic	Hours	Teaching Method
1	Introduction to ASIC design and programmable logic device.	4	Lecture
2	Introduction to VHDL.	2	Lecture
	Lab 1. Implementation of Basic operation of ALU using ISE Design Suite.	2	Lab
3	Using Port Map.	2	Lecture
	Lab 2. Building Full-adder Using Port Map.	2	Lab
4	Combinational Statement.	2	Lecture
	Lab 3. Digital Display and Gray Codes.	2	Lab
5	Sequential Statement.	2	Lecture
	Lab 4. Accumulator; Multiplier Using Booth Algorithm.	2	Lab
6	Lab 5. 12-hour Clock.	4	Lab
7	Finite State Machine.	2	Lecture
	Lab 6. Random number generator and Multi-cycle Processor Control.	2	Lab
8	Lab 7. Register file and memory.	4	Lab
9-10	Lab 8. Single Cycle Processor.	8	Lab
11	Lab 9. FFT Implementation.	4	Lab
12-14	Lab 10. Single Cycle Processor.	12	Lab
15	Summary and Exam.	4	Lecture

Computer Design

Instruction Set Design
- Machine Language
- Compiler View
- "Computer Architecture"
- "Instruction Set Processor"

"Building Architect"

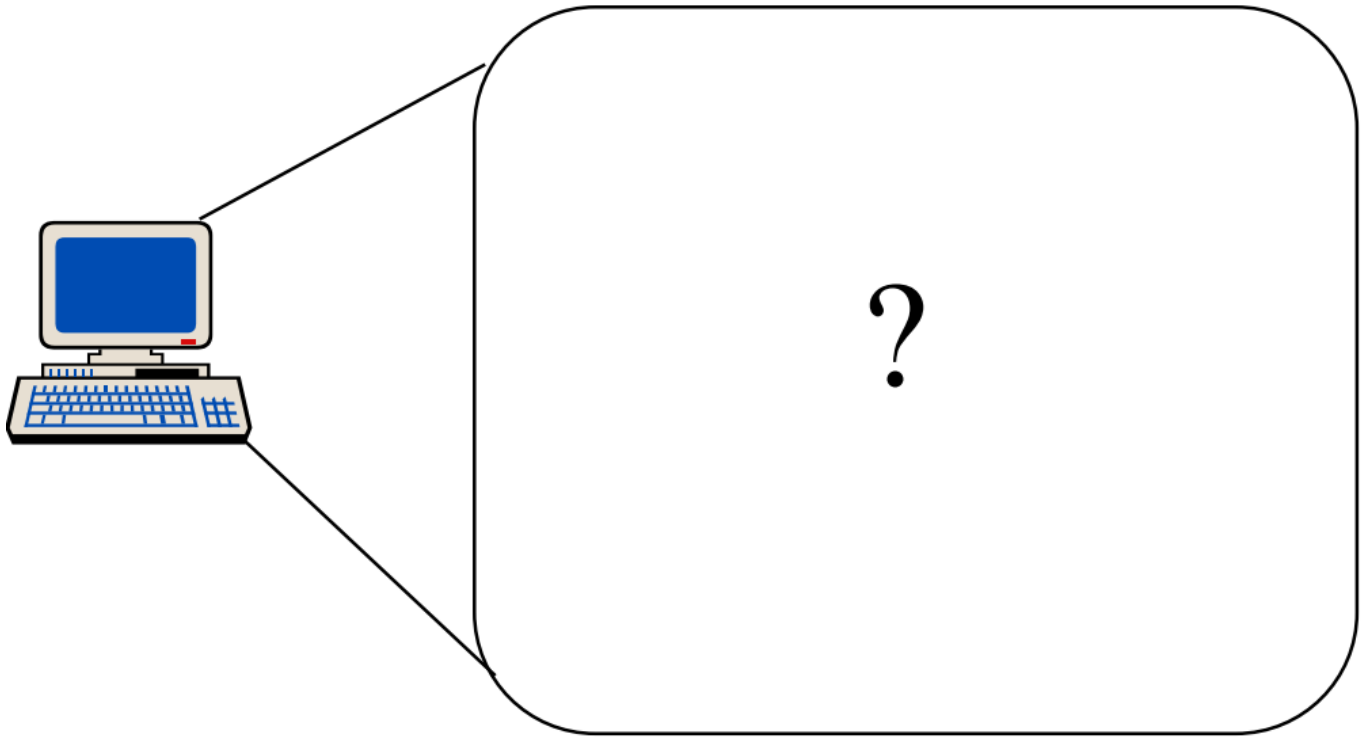
Computer Hardware Design
- Machine Implementation
- Logic Designer's View
- "Processor Architecture"
- "Computer Organization"

“Construction Engineer”

Few people design computers! Very few design instruction sets!
Many people design computer components.
Very many people are concerned with computer function, in detail.

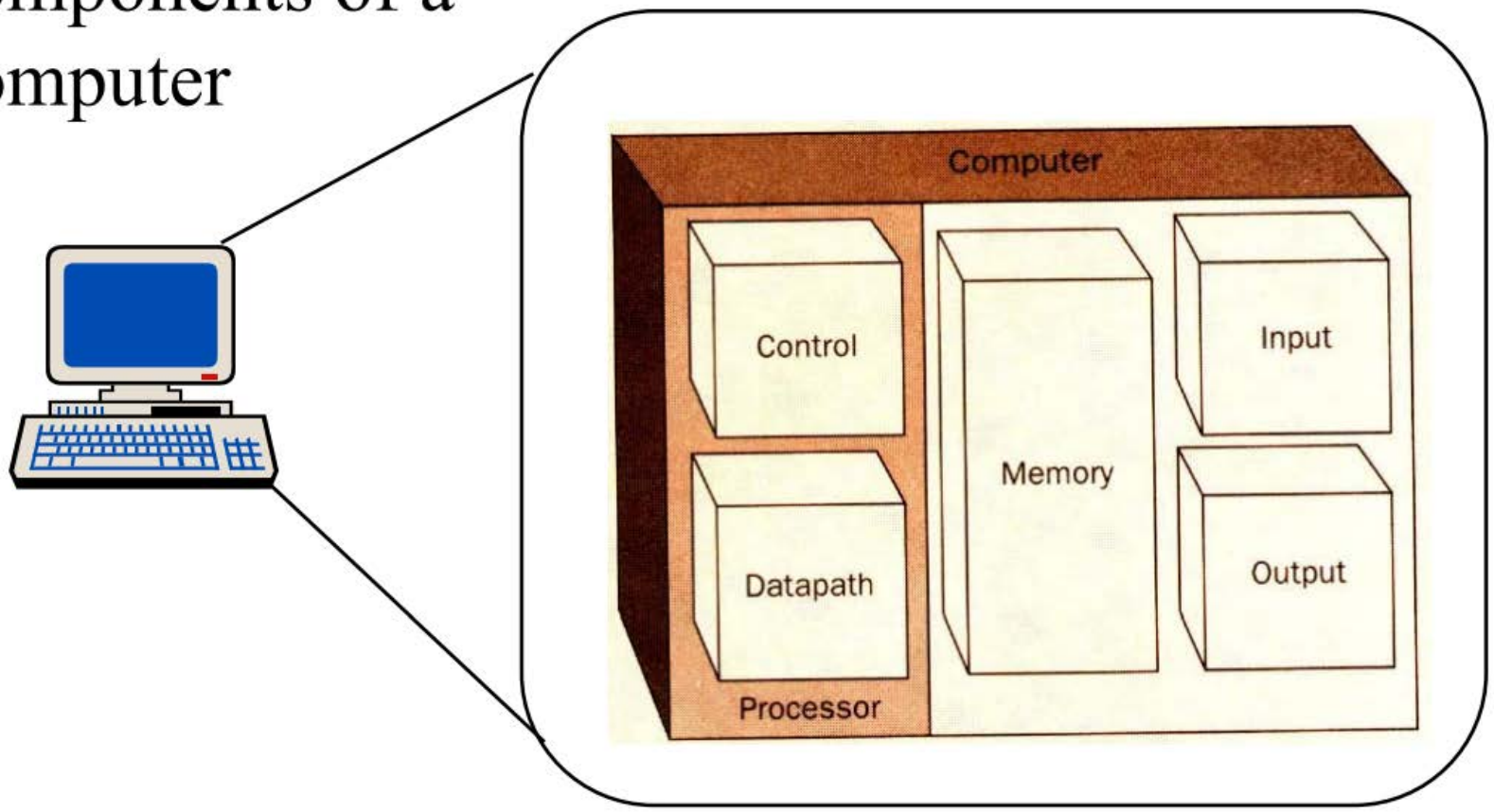
The Big Picture

- What is inside a computer?
- How does it execute my program?

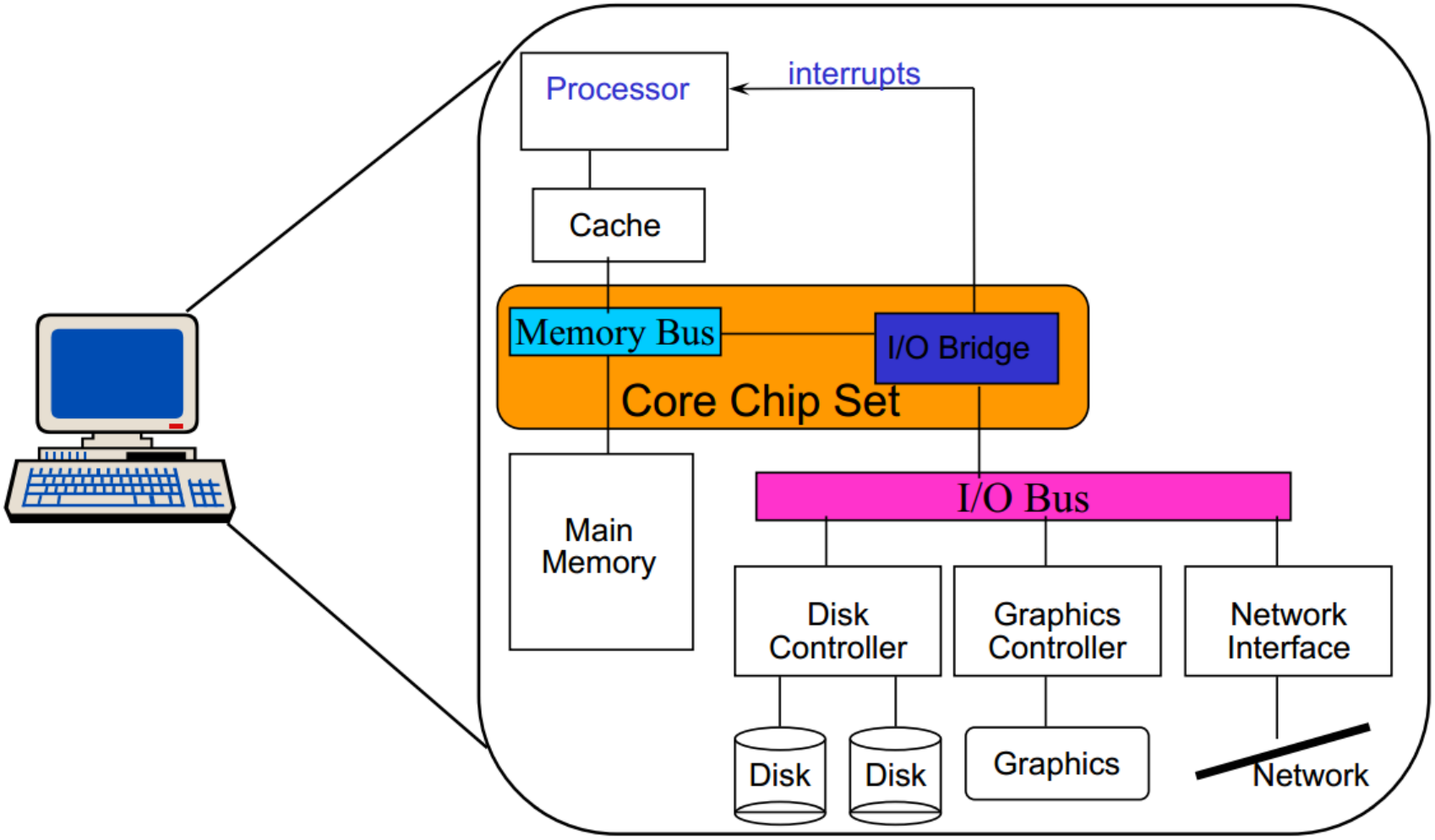


The Big Picture

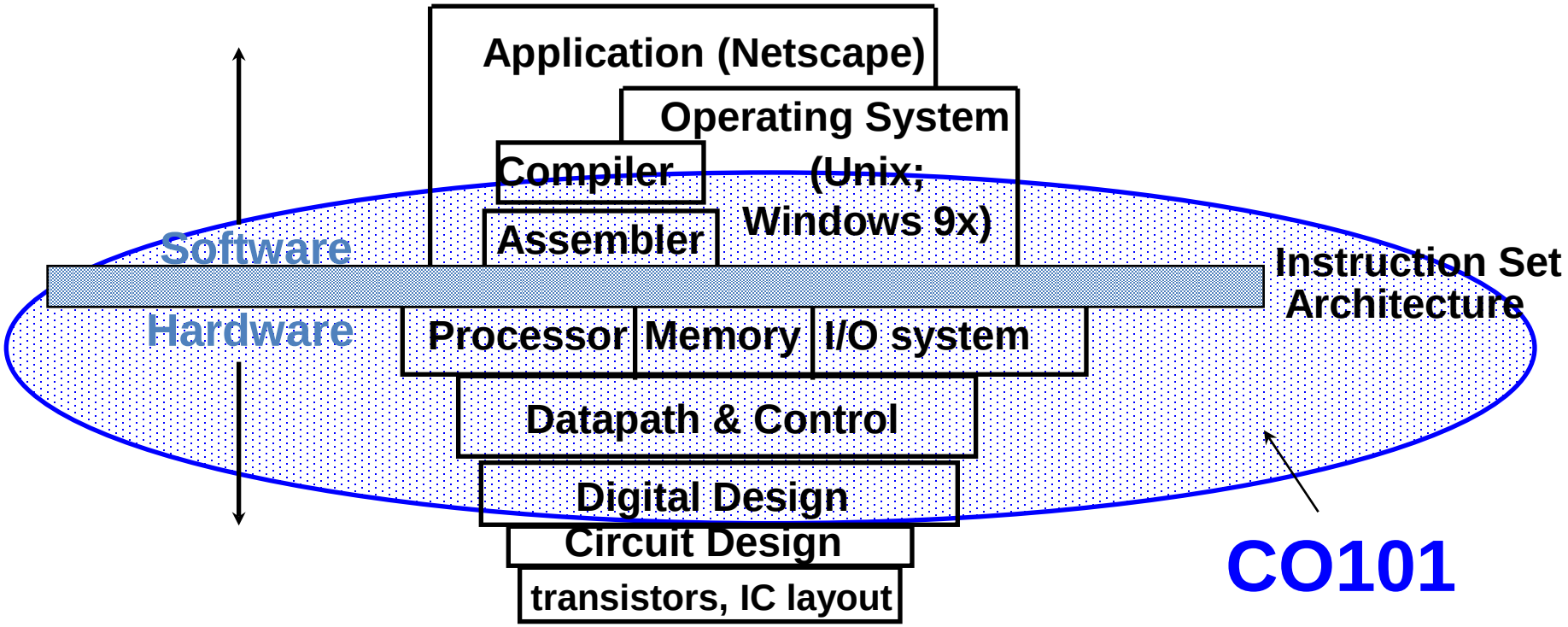
- The Five Classic Components of a Computer



Computer Organization



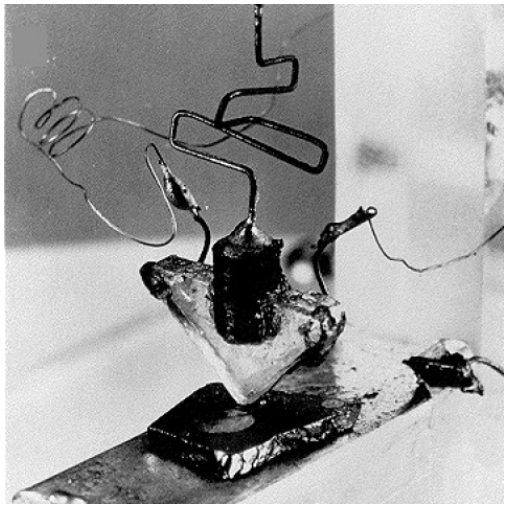
Computer Organization



- Key Idea: levels of abstraction

The Evolution of Computer Hardware

- When was the first transistor invented?

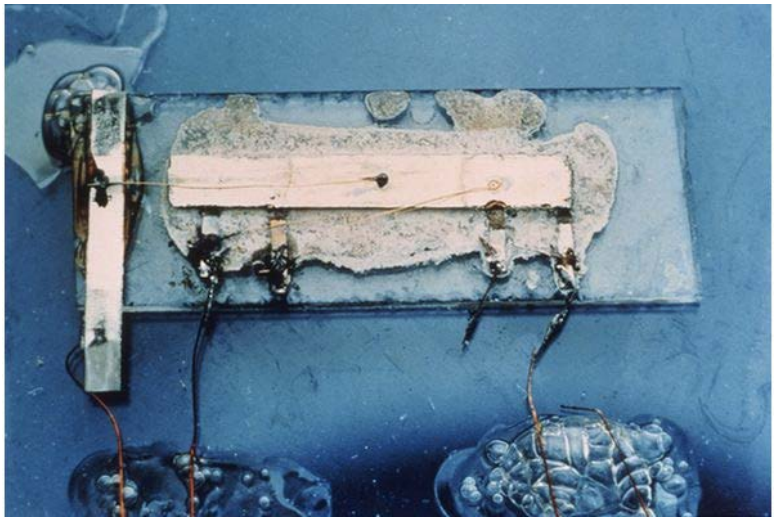


1947 - the bi-polar transistor – by Bardeen *et.al* at Bell Laboratories

UNIVAC I (Universal Automatic Computer) – the first commercial computer in USA

The Evolution of Computer Hardware

- When was the first IC (integrated circuit) invented?

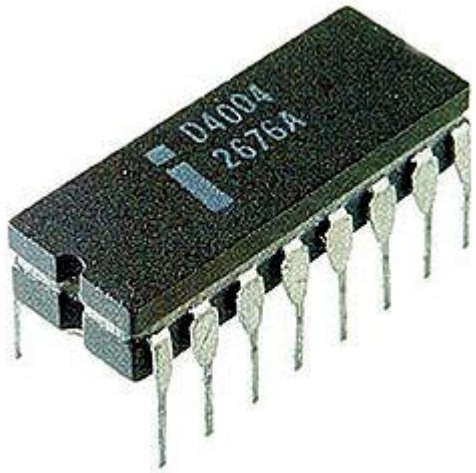


1958, by Jack Kilby@Texas Instruments, by hand, several transistors, resistors and capacitors on a single substrate.

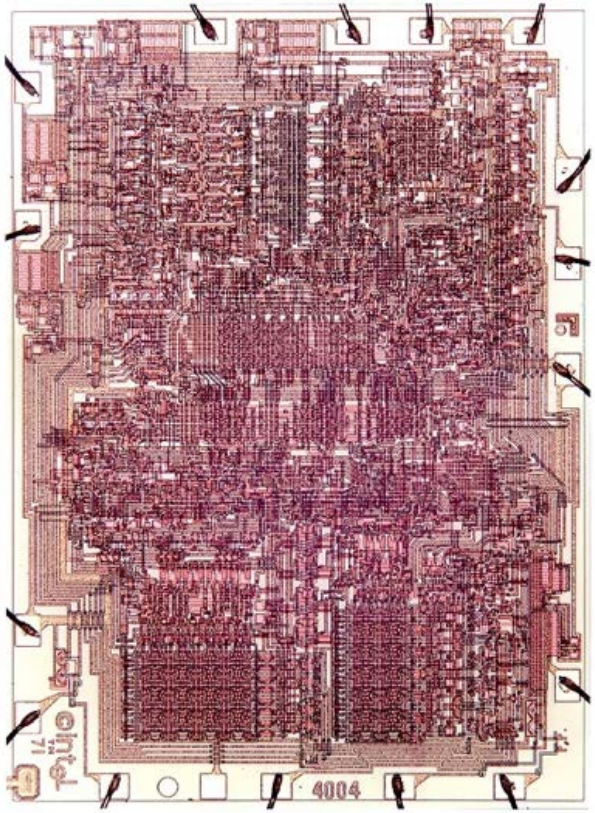
IBM System/360, 2MHz, 128KB ~ 256KB

The Evolution of Computer Hardware

- When was the first Microprocessor?

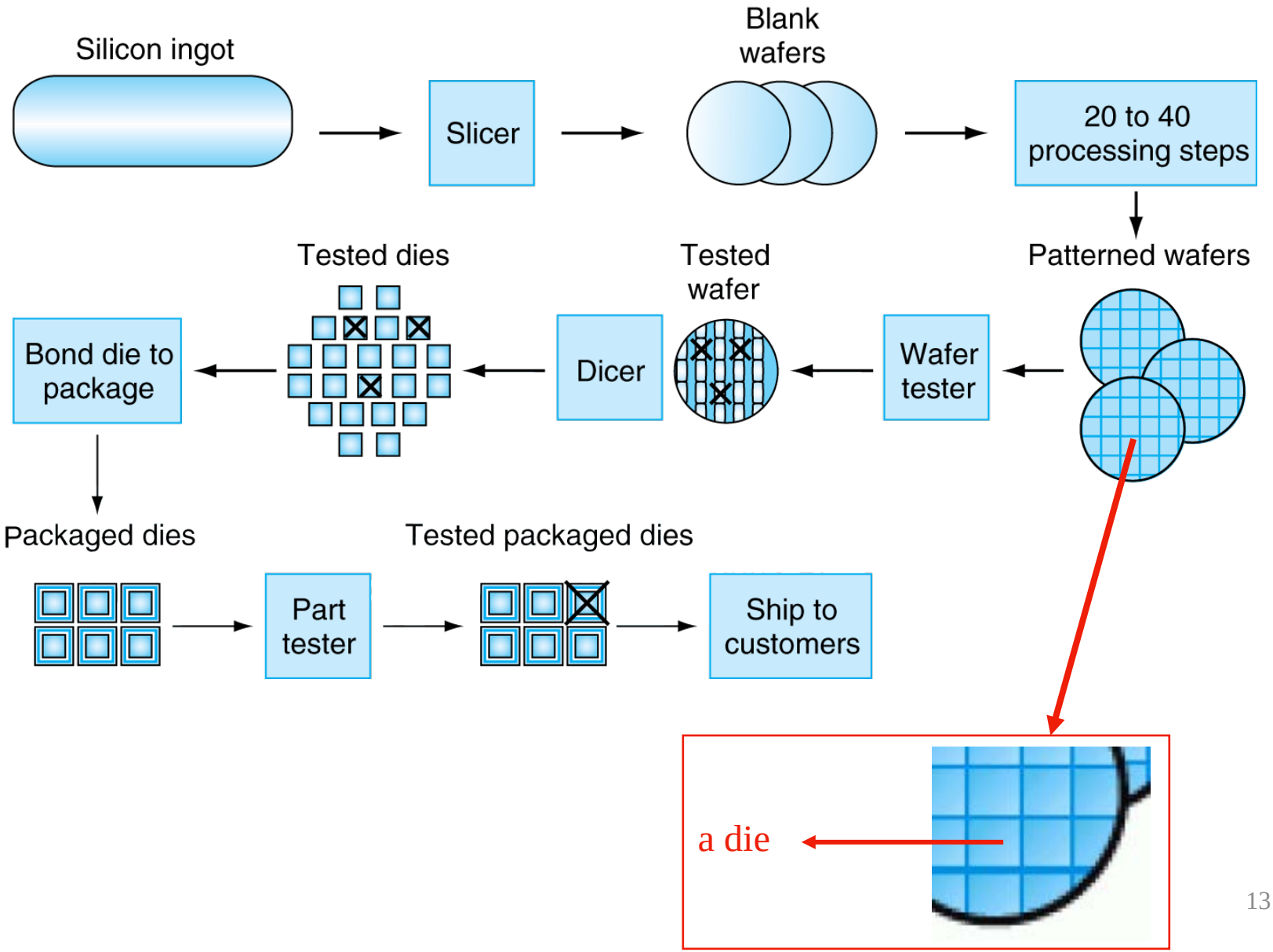


1971, Intel 4004



The History of Intel Processors: https://www.youtube.com/watch?v=Qu2njWY3Hjk

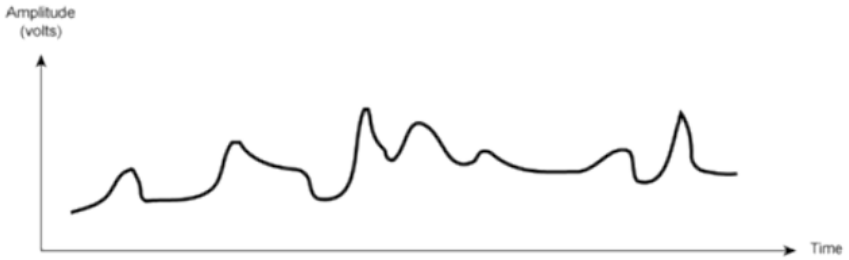
The Chip Manufacturing Process

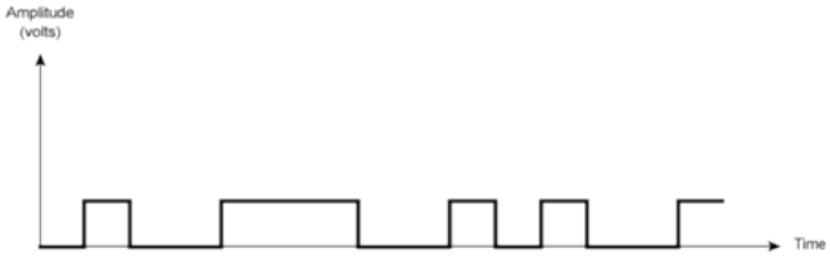


Analog vs. Digital

- Analog Signal
  - Vary in a smooth way over time.
  - Analog data are continuous valued
    - Example: audio, video

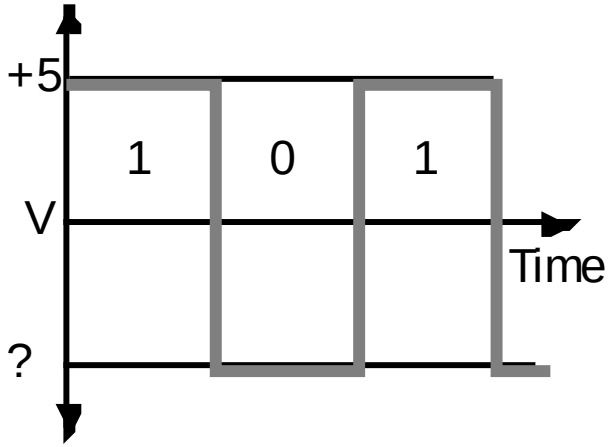
- Digital Signal
  - Maintains a constant level then changes to another constant level (generally operate in one of the two states).
  - Digital data are discrete value.
    - Example: computer data.

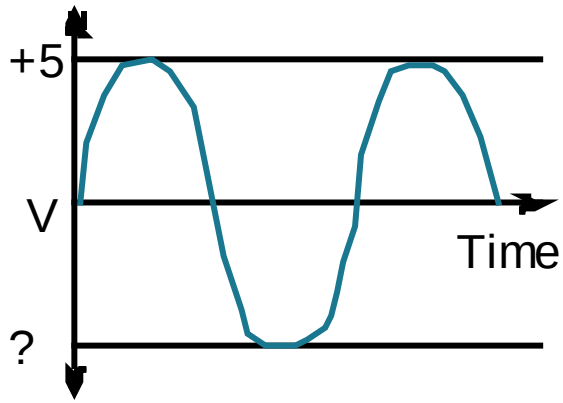




Digital Systems

Digital vs. Analog Waveforms





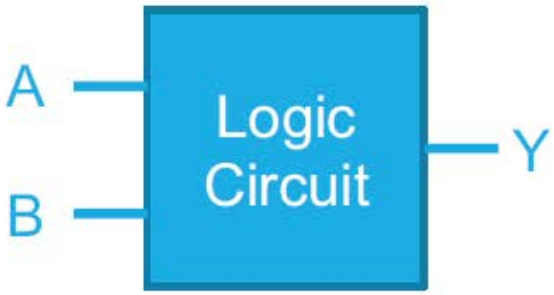
Digital:
only assumes discrete values

Analog:
values vary over a broad range continuously

Truth Table

- A means for describing how a logic circuit's output depends on the logic levels present at the circuit's inputs.

- The number of input combinations will equal 2N for an N-input truth table.



Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Digital Hardware Systems

Boolean Algebra and Logical Operators

Algebra: **variables, values, operations**

In Boolean algebra, the values are the symbols 0 and 1
If a logic statement is false, it has value 0
If a logic statement is true, it has value 1

Operations: AND, OR, NOT

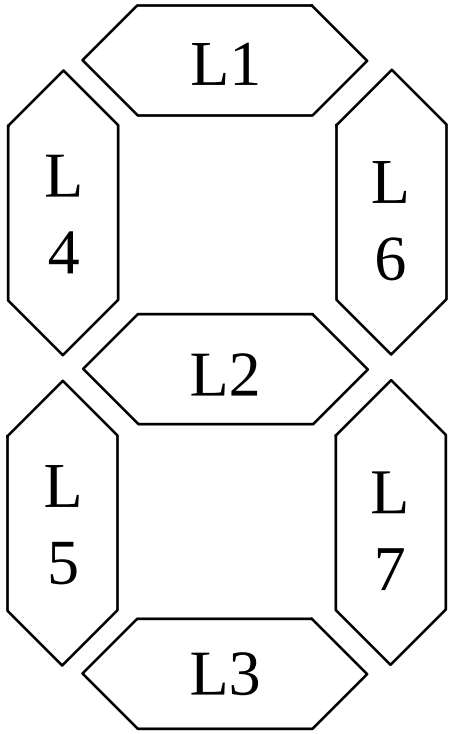
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

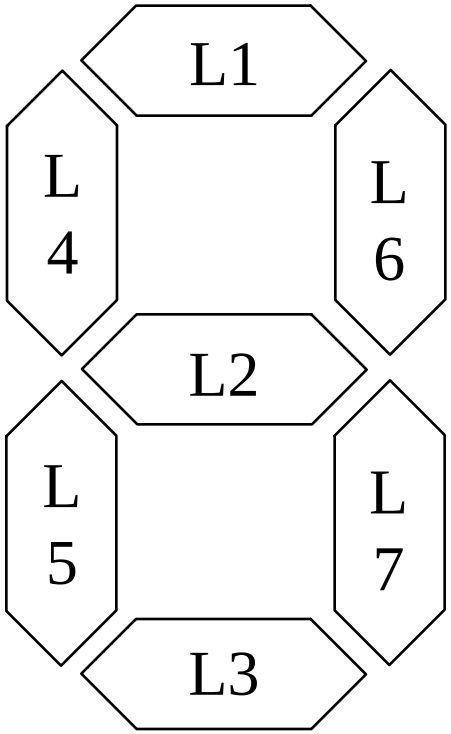
Case Study of a Simple Logic Design:
Seven Segment Display

- Chip to drive digital display

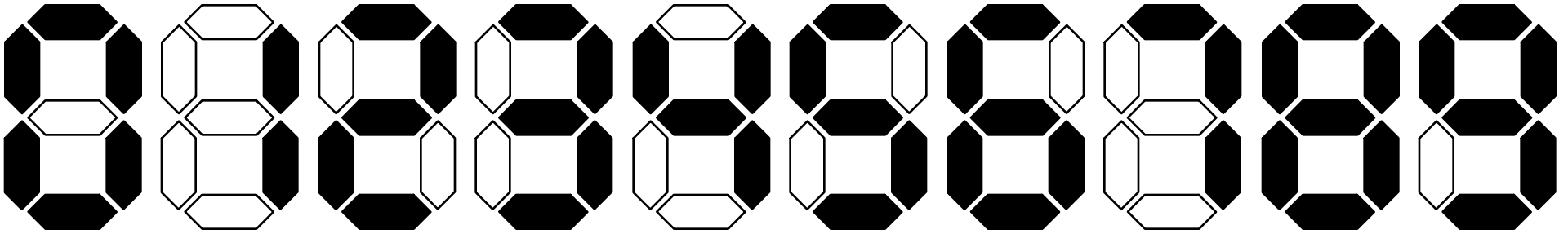


B3	B2	B1	B0	Val
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Case Study (cont.)

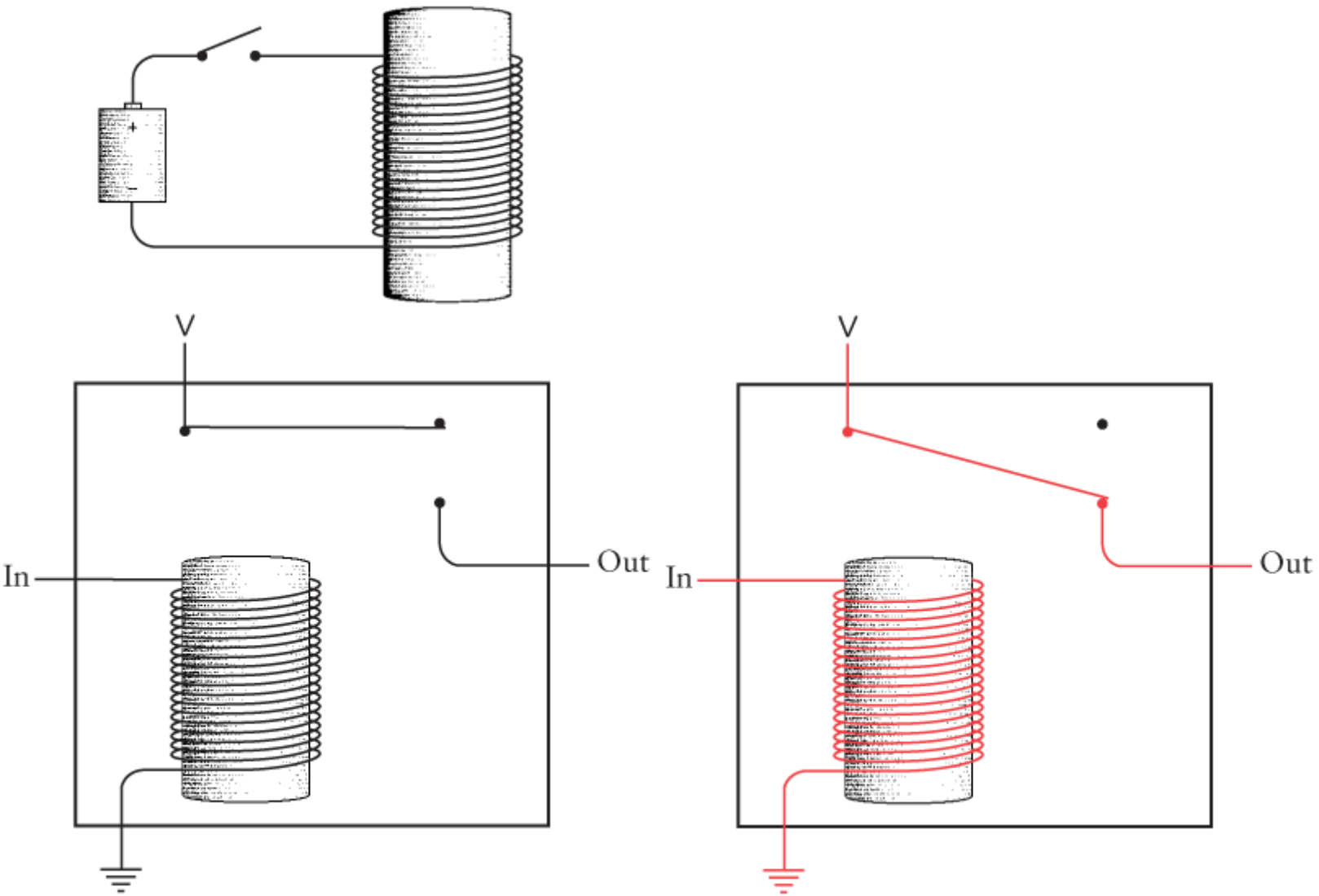


B3	B2	B1	B0	Val	L1	L2	L3	L4	L5	L6	L7
0	0	0	0	0	1	0	1	1	1	1	1
0	0	0	1	1	0	0	0	0	0	1	1
0	0	1	0	2	1	1	1	0	1	1	0
0	0	1	1	3	1	1	1	0	0	1	1
0	1	0	0	4	0	1	0	1	0	1	1
0	1	0	1	5	1	1	1	1	0	0	1
0	1	1	0	6	1	1	1	1	1	0	1
0	1	1	1	7	1	0	0	0	0	1	1
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	1	0	1	1

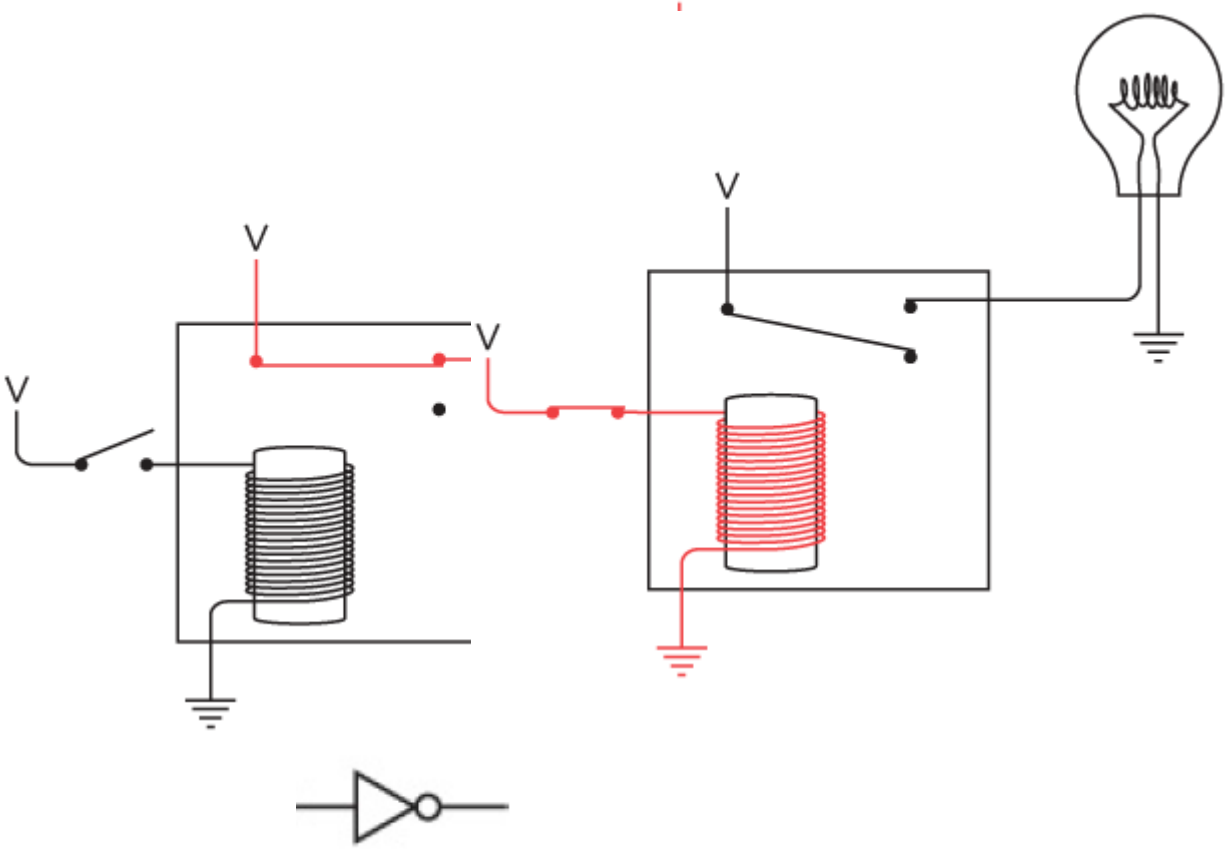


How do computers work?

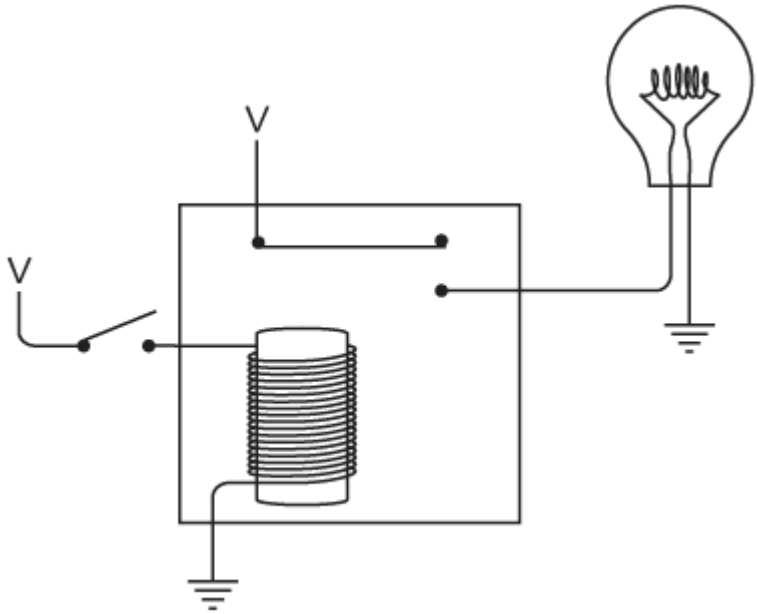
The basic unit: repeater

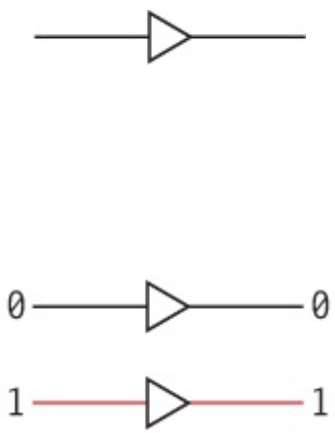


Basic logic unit: NOT

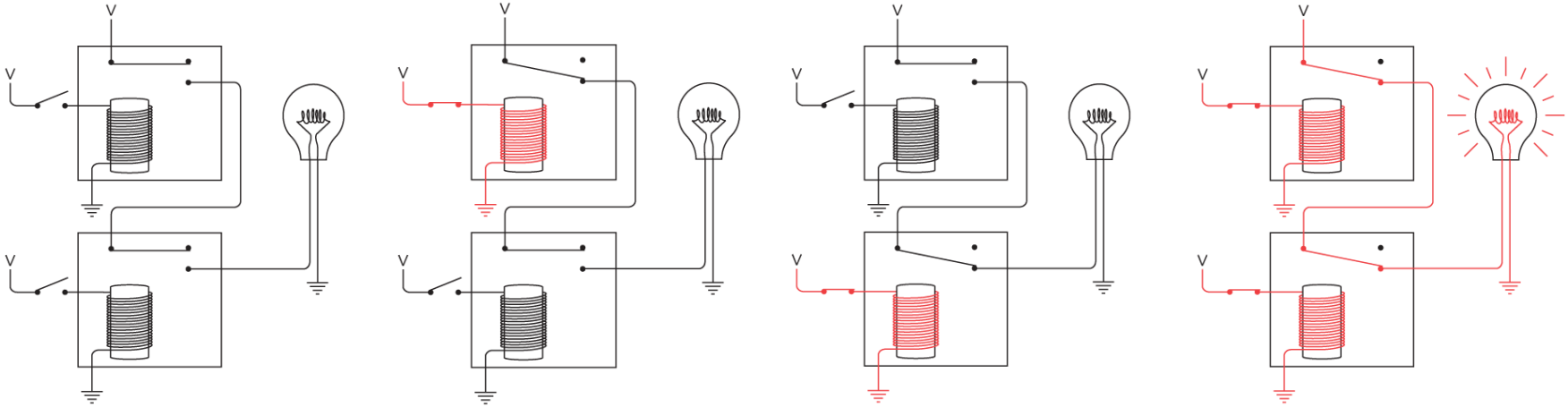


Basic logic unit: buffer

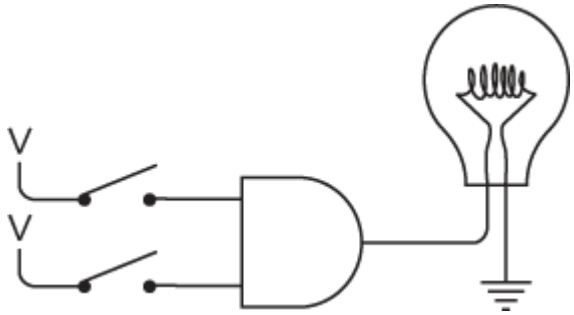


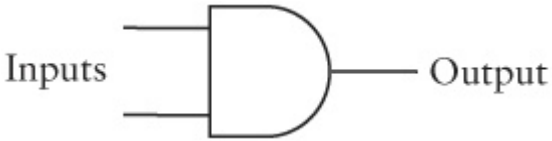


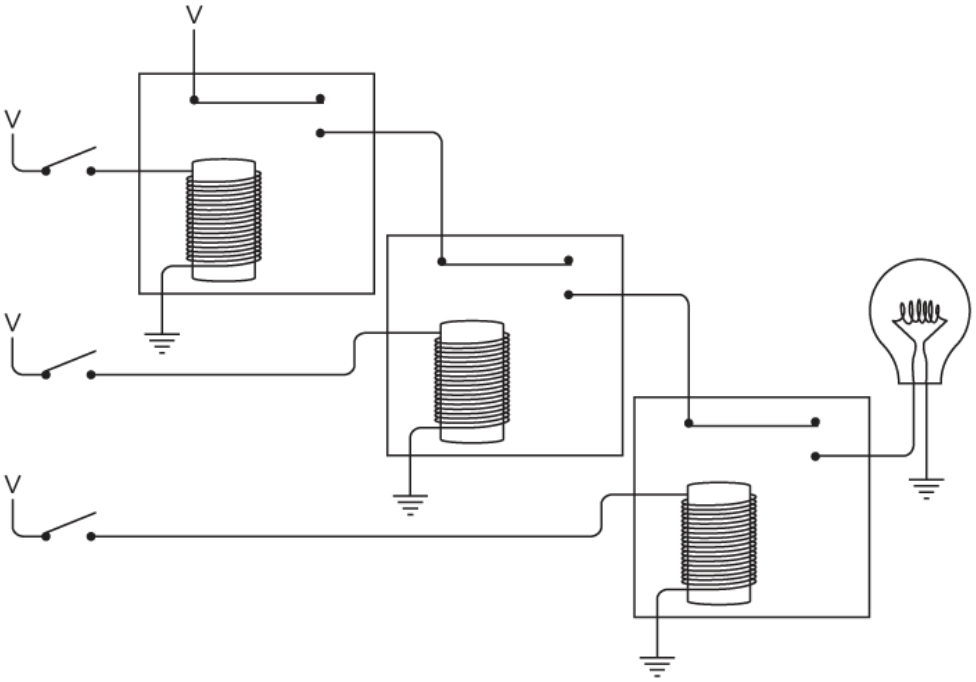
Basic logic unit: AND

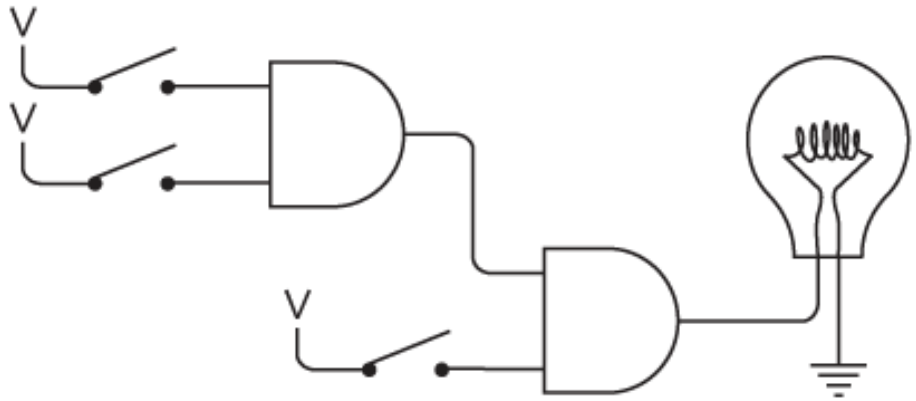


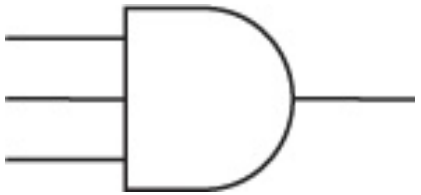
AND	0	1
0	0	0
1	0	1



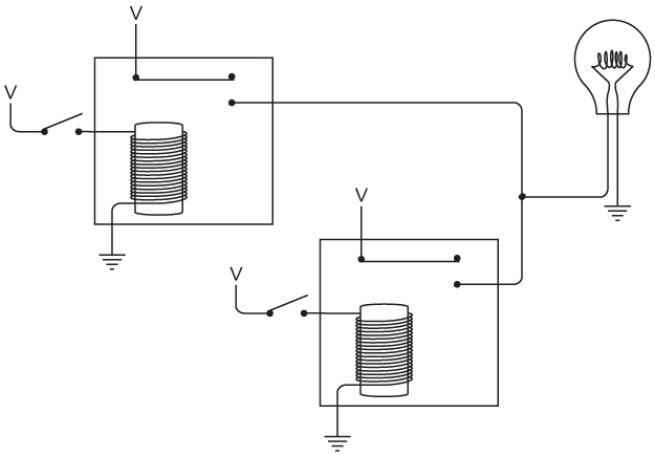


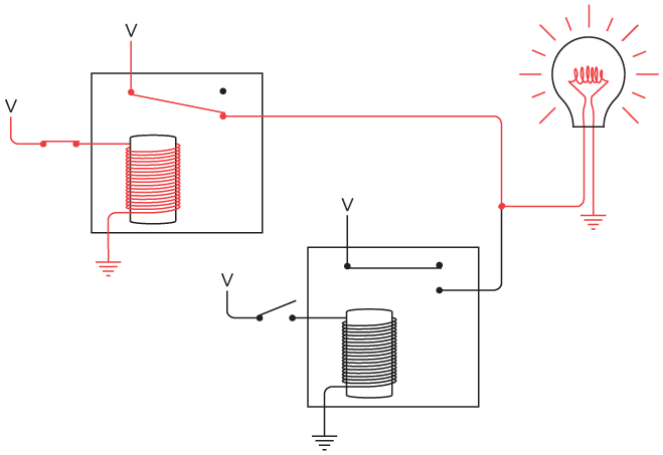

V
V
V
V

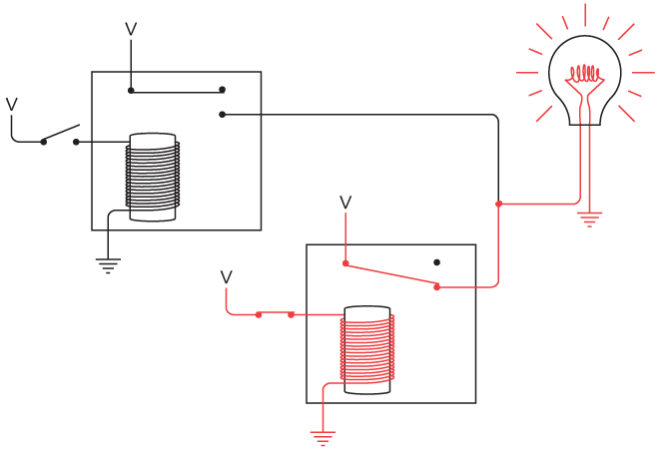

V
V
V

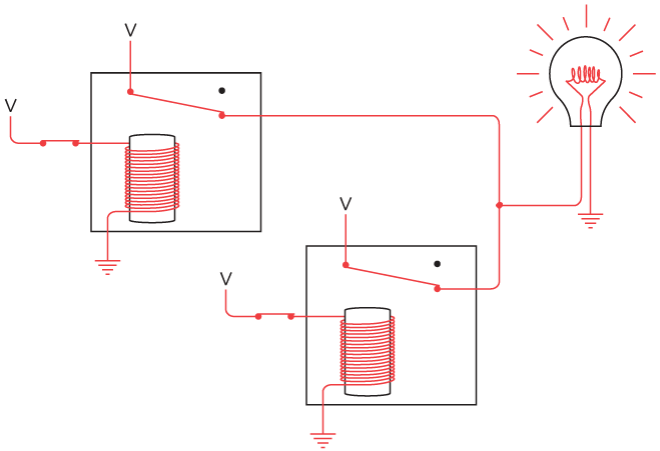


Basic logic unit: OR

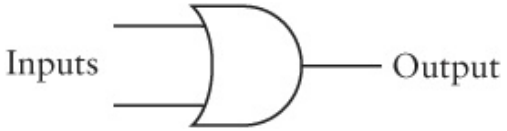




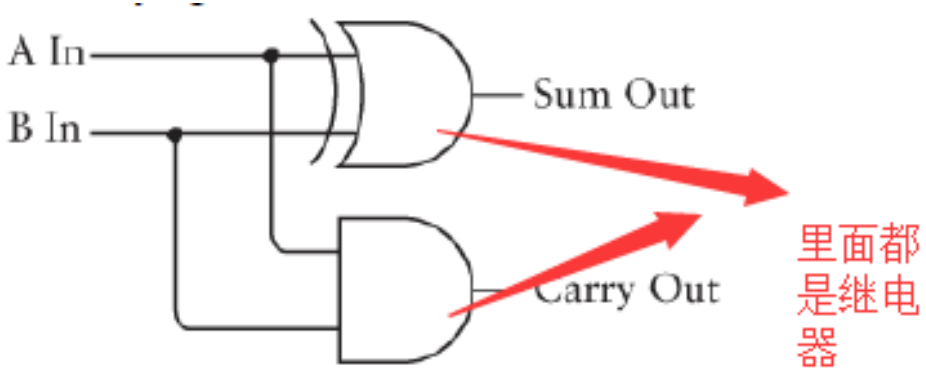


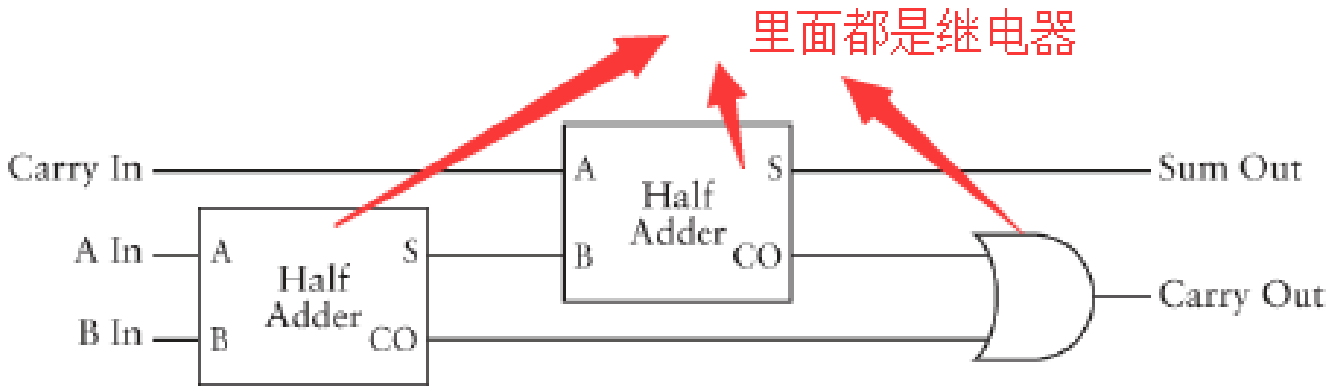


OR	0	1
0	0	1
1	1	1



Half-adder and Full-adder





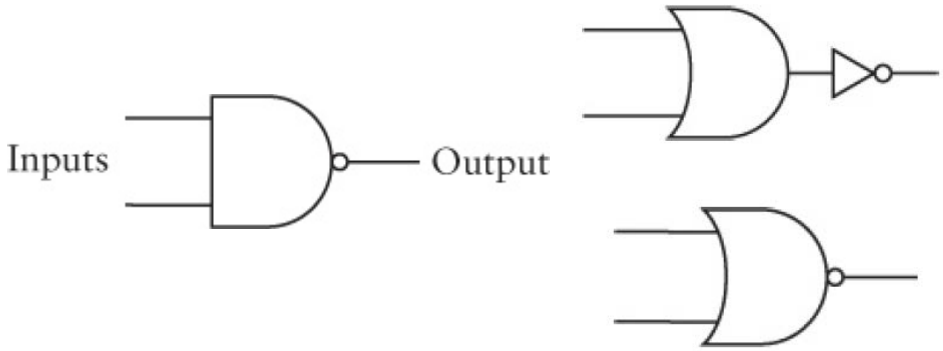
Basic logic unit: NAND and NOR

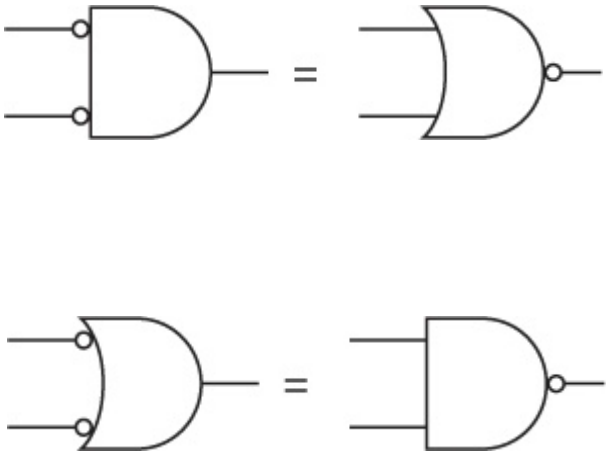
AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

NAND	0	1
0	1	1
1	1	0

NOR	0	1
0	1	0
1	0	0

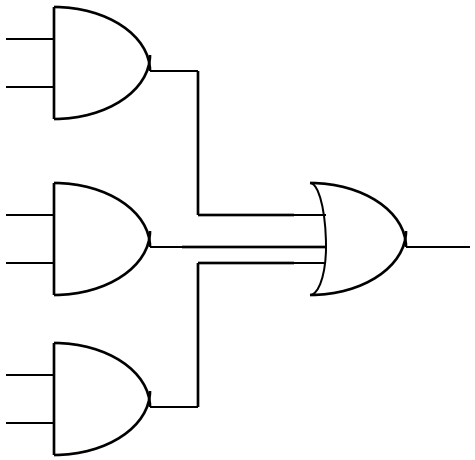




Case Study

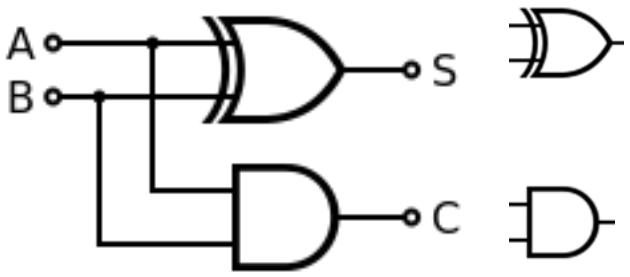
- Implement L4:

B3	B2	B1	B0	L4
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1



Some gate level implementation of the Boolean function for L4

Building a 1-bit Binary Half Adder



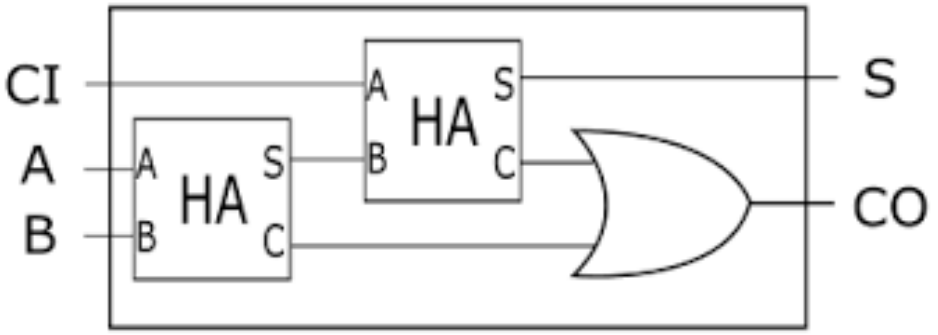
XOR gate

AND gate

Inputs		Outputs	
A	B	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

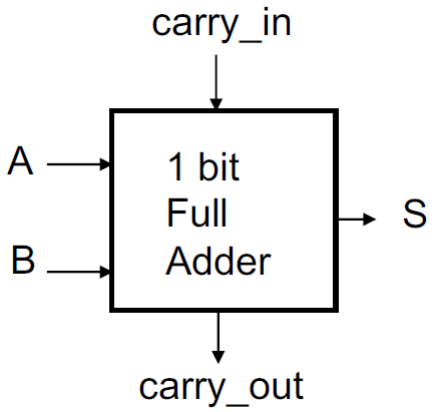
- A: input
- B: input
- S: output
- C: output

- Using two half adders t



Building a 1-bit Binary Full Adder

A	B	carry_in	carry_out	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



S = A xor B xor carry_in

carry_out = A&B | A&carry_in | B&carry_in

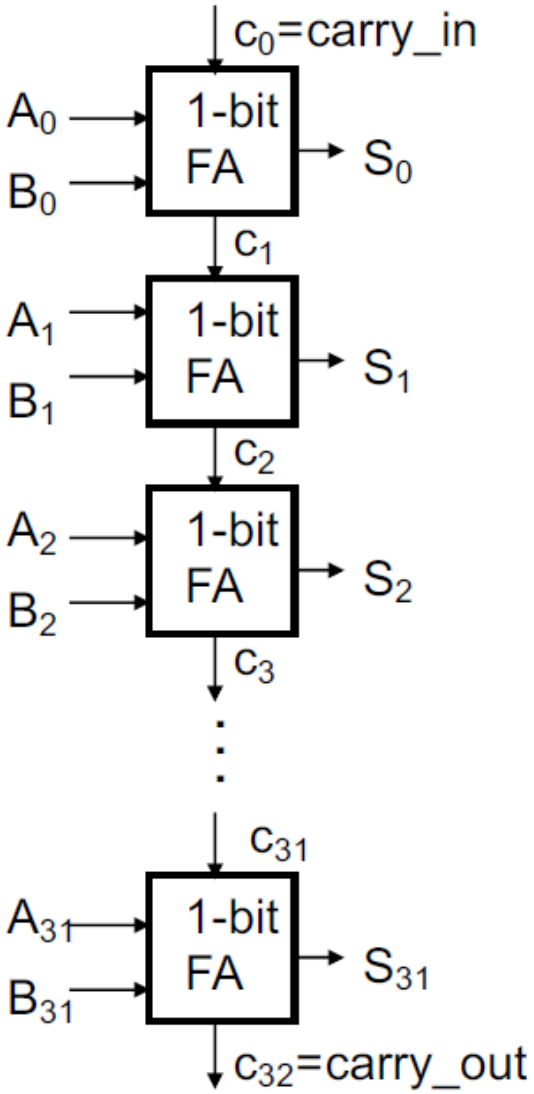
(majority function)

How can we use it to build a 32-bit adder?

How can we modify it easily to build an adder/subtractor?

Building 32-bit Adder

- Jus
  to
  co
- Rip
  -
  -



-out of the least significant bit FA
ext least significant bit and

CA)

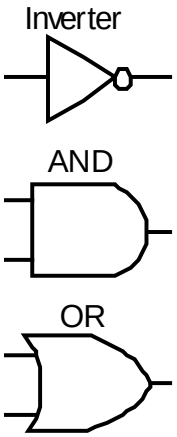
c, so small (low cost).

d lots of glitching (so lots of energy

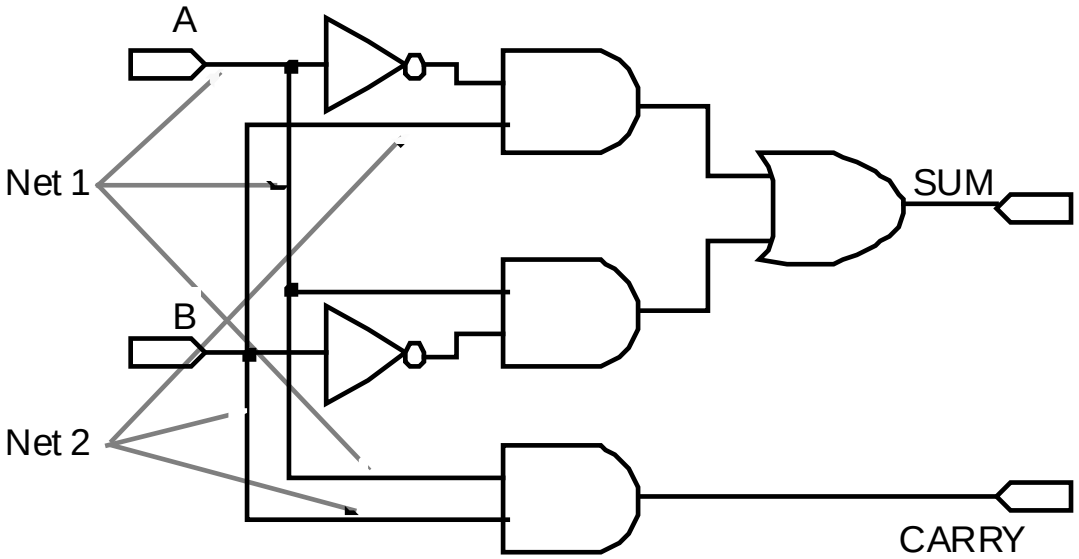
Gate Representations of a Digital Design

most widely used primitive building block in digital system design

Standard Logic Gate Representation



Half Adder Schematic

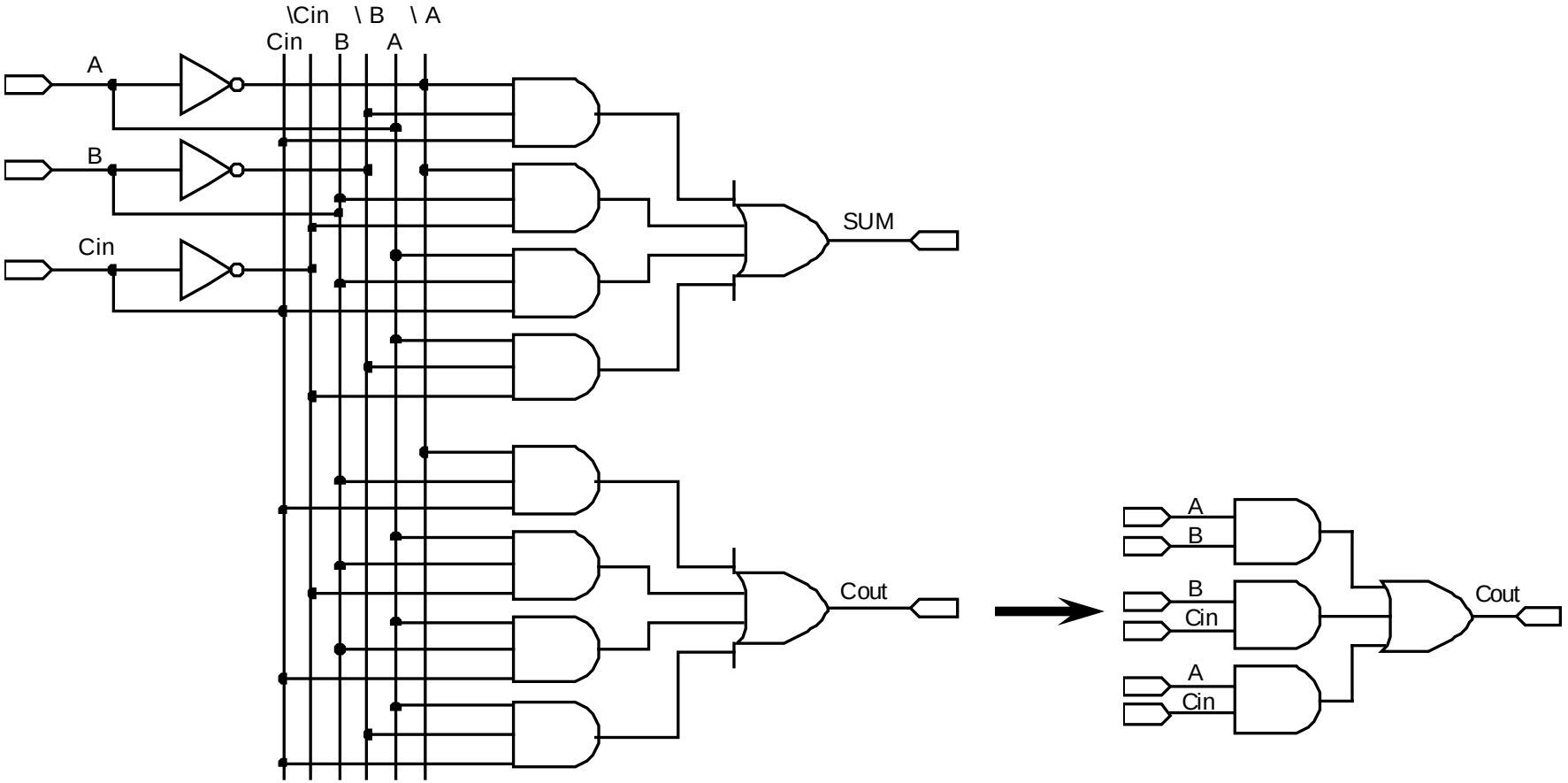


Net: **electrically connected collection of wires**

Netlist: **tabulation of gate inputs & outputs and the nets they are connected to**

Representations of a Digital Design: Gates

Full Adder Schematic



Fan-in: **number of inputs to a gate**
Fan-out: **number of gate inputs an output is connected to**

Technology "Rules of Composition" place limits on fan-in/fan-out

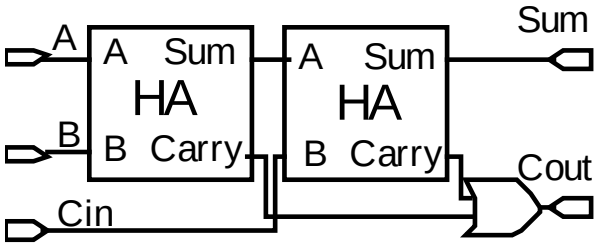
Block Representation of a Digital Design

structural organization of the design

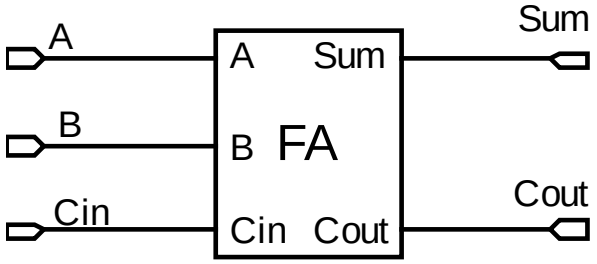
black boxes with input and output connections

corresponds to well defined functions

concentrates on how the components are composed by wiring



Full Adder realized in terms of composition of half adder blocks

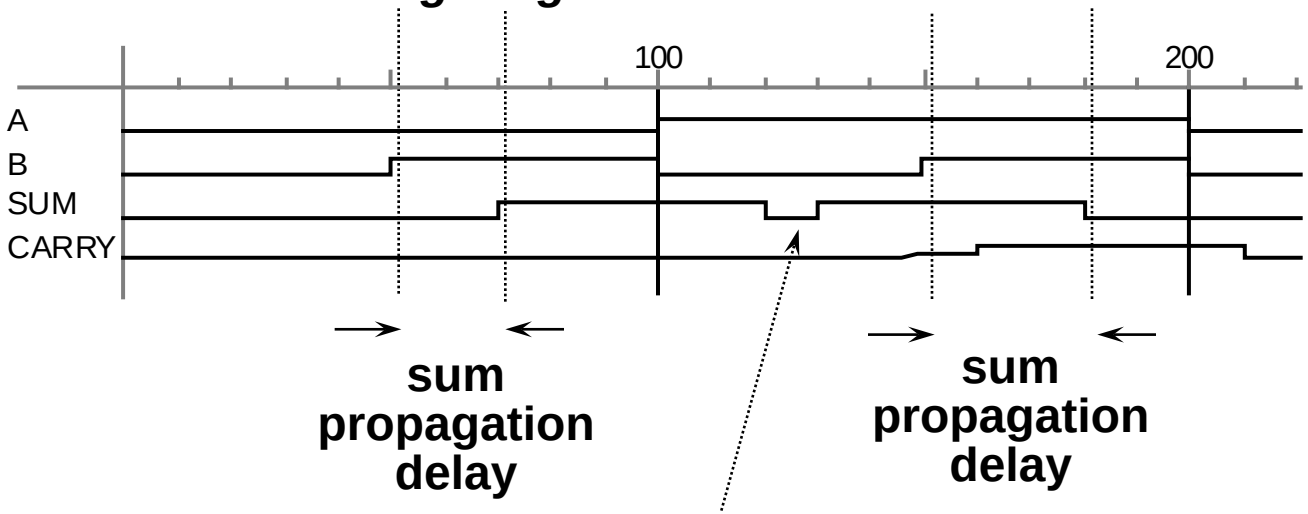


Block diagram representation of the Full Adder

Waveform Representation

dynamic behavior of a circuit
real circuits have non-zero delays

Timing Diagram of the Half Adder

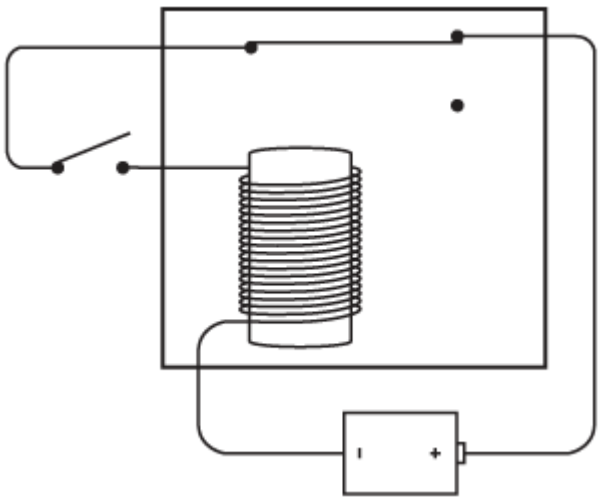


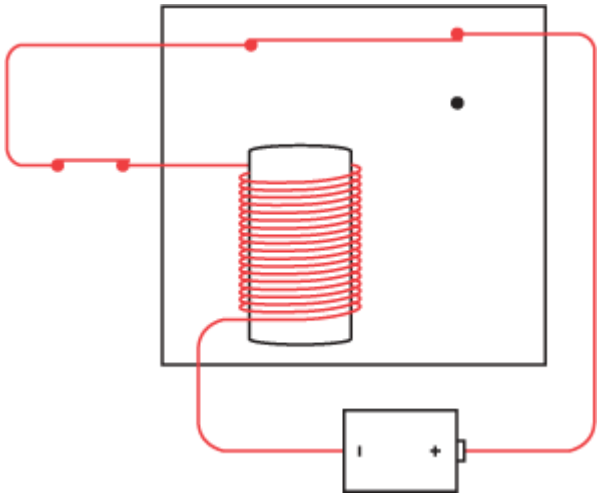
Output changes are delayed from input changes

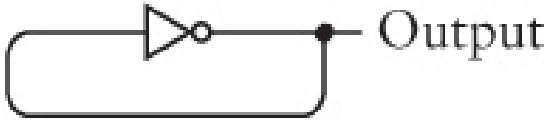
The propagation delay is sensitive to paths in the circuit

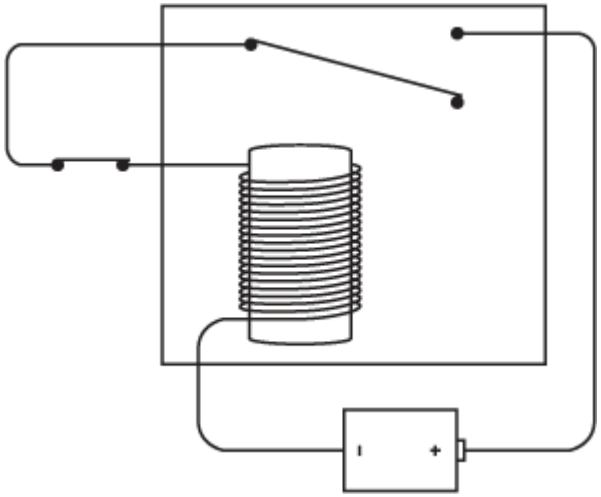
Outputs may temporarily change from the correct value to the wrong value back again to the correct value: this is called a *glitch* or *hazard*

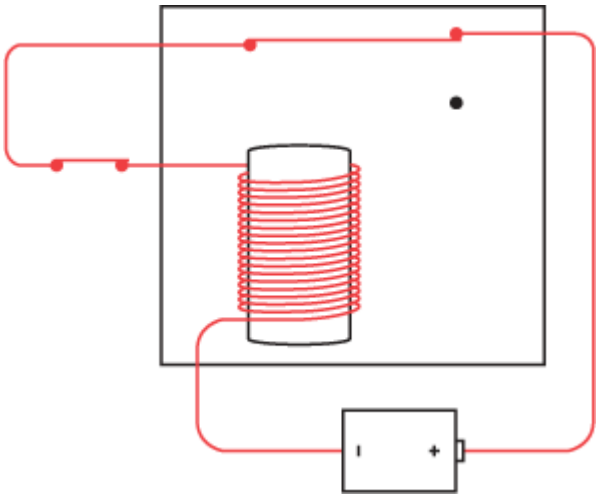
Feedback and Flip-Flops

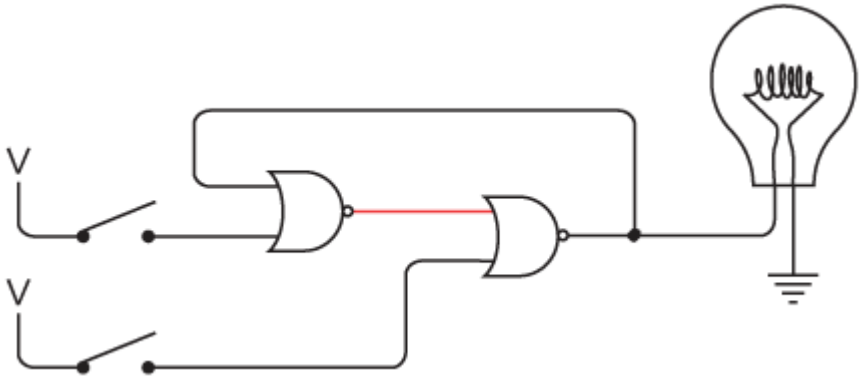


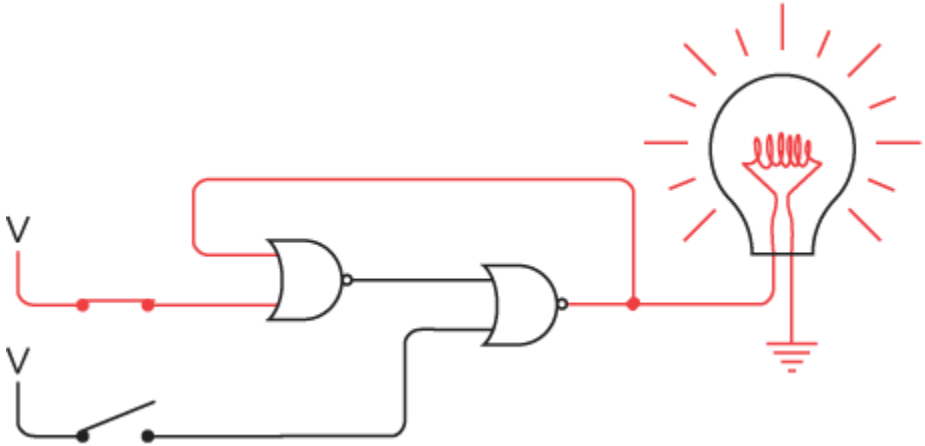


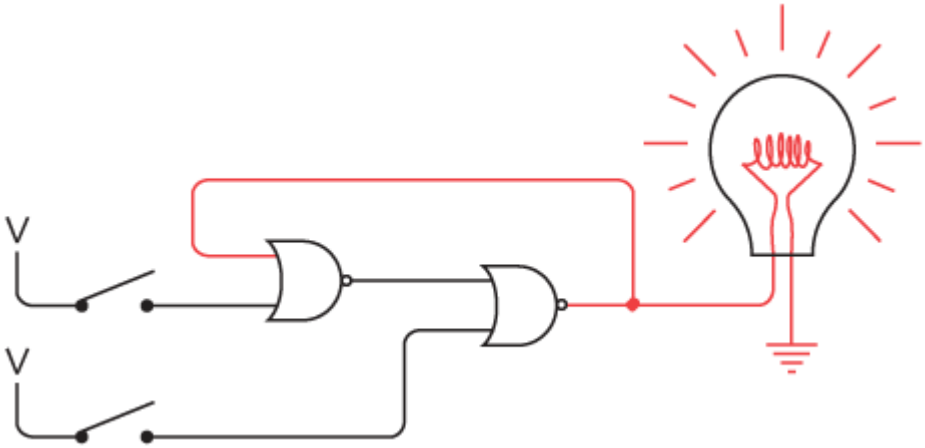


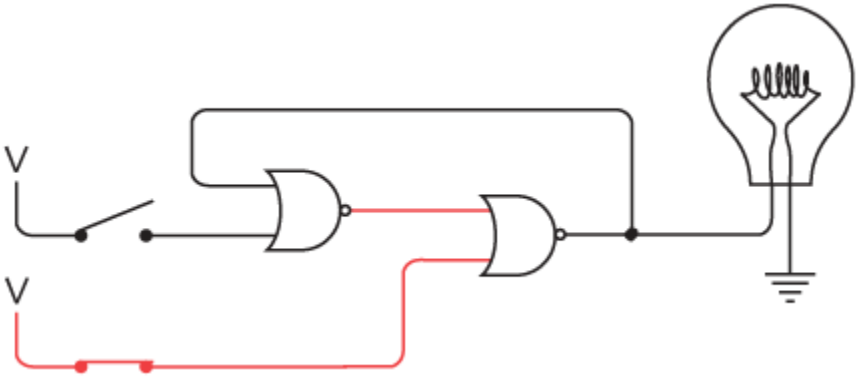




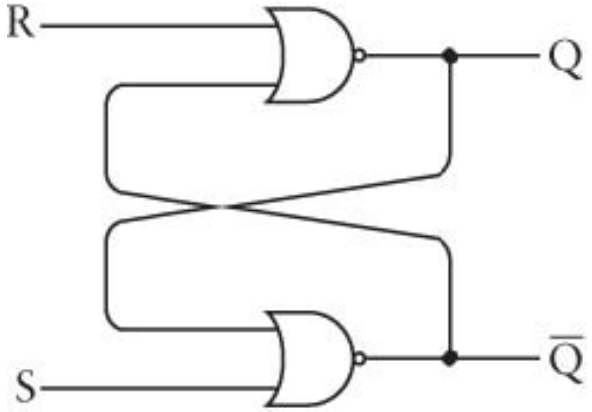

V
V

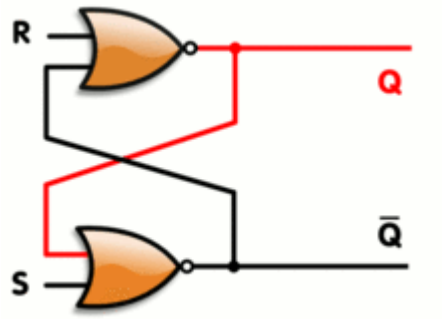

V
V

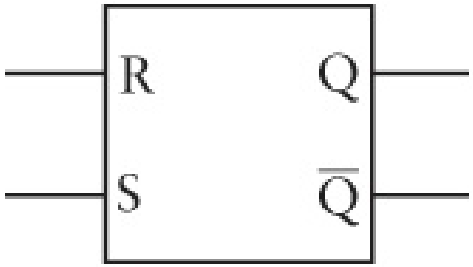

V
V


V
V

Inputs		Outputs	
S	R	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	Q	$\overline{Q}$
1	1	Disallowd	

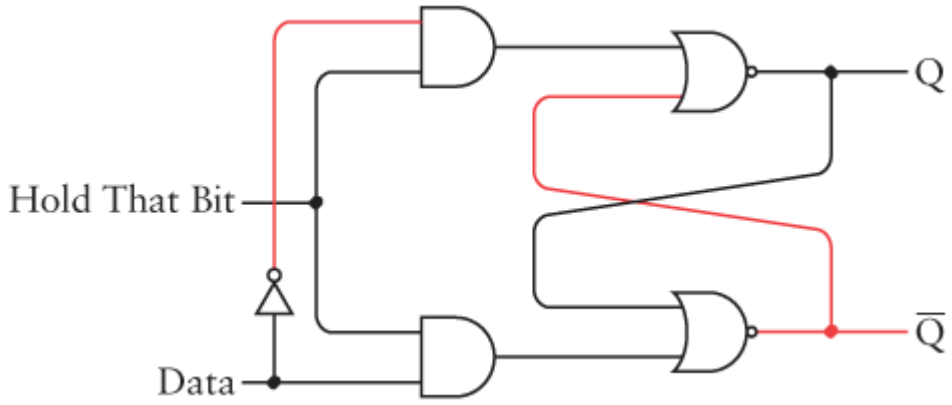


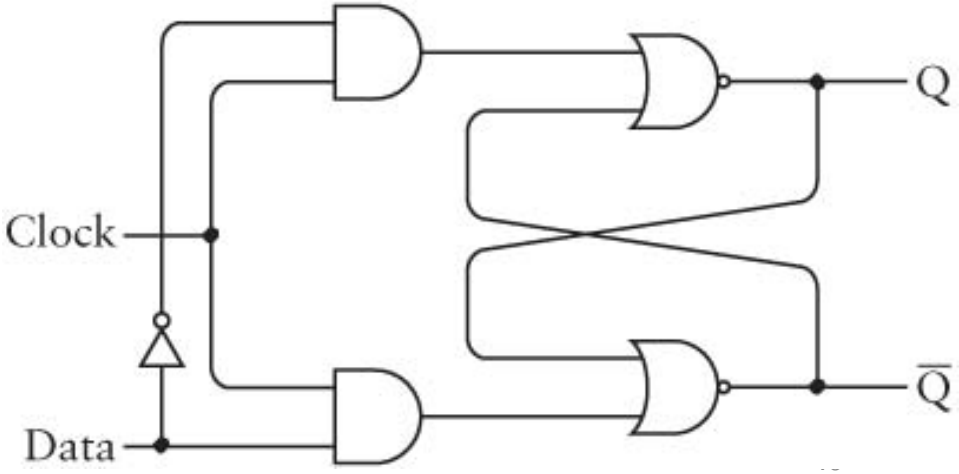




Inputs		Outputs
Data	**Hold That Bit**	**Q**
0	1	0
1	1	1
0	0	Q
1	0	Q

Inputs		Outputs	
D	**Clk**	**Q**	**Q-bar**
0	1	0	1
1	1	1	0
X	0	Q	Q-bar

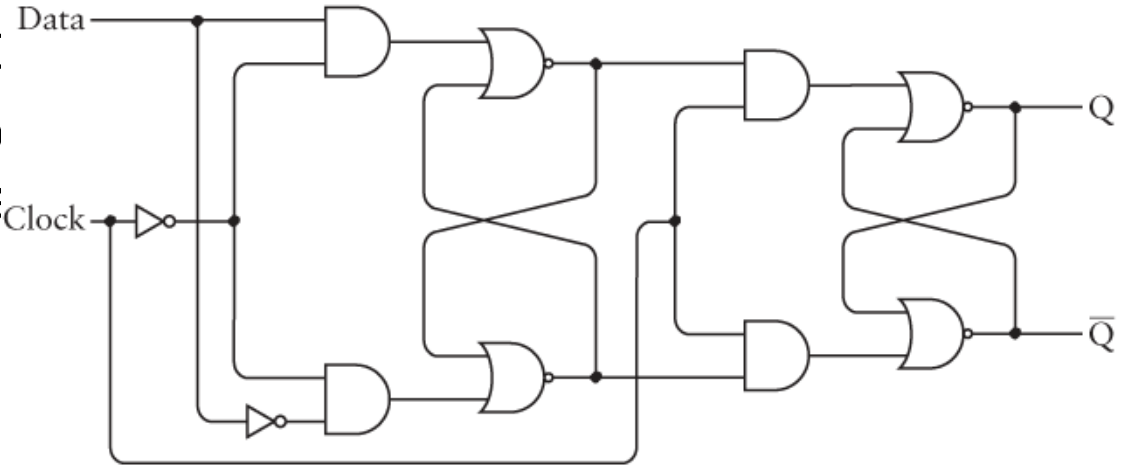


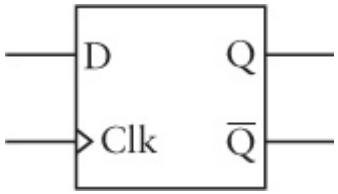


An edge-triggered D-type flip-flop

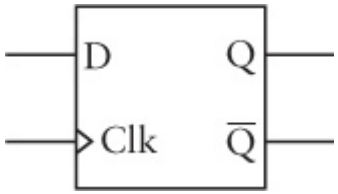
- The idea here is that the Clock input controls both the first stage and the second stage. But notice that the clock is inverted in the first stage. This means that the first stage works exactly like a D-type flip-flop except that the Data input is stored when the Clock is 0. The outputs

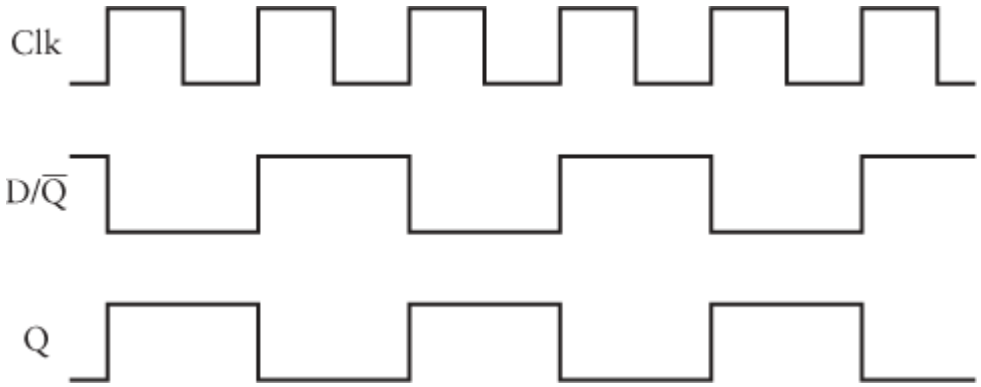
Inputs		Outputs	
D	Clk	Q	$\overline{Q}$
1	0	0	1
1	↑	1	0
0	1	1	0
0	0	1	0
0	↑	0	1
1	1	0	1





An edge-triggered D-type flip-flop



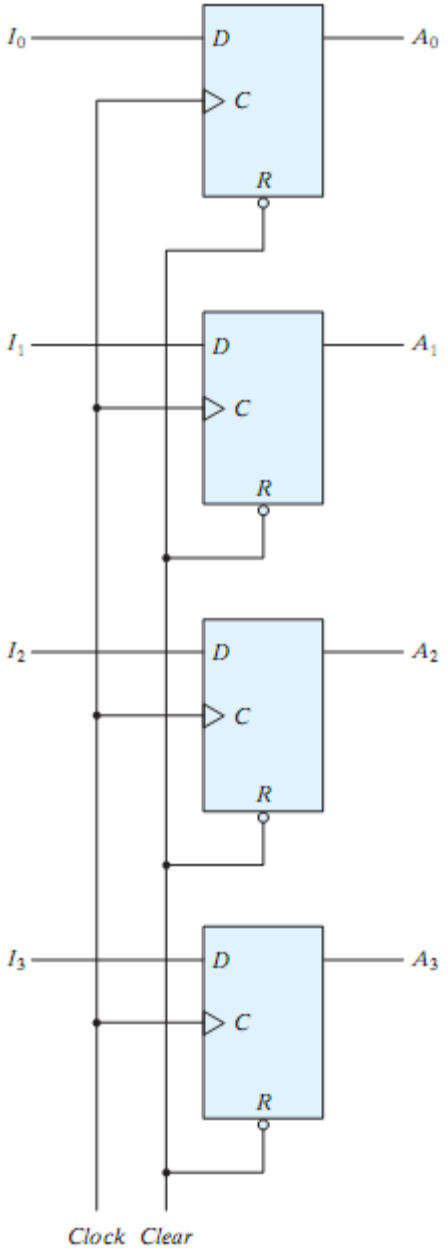


Registers

- A register is a group of flip-flops.
- An n-bit register is made of n flip-flops and can store n bits.
- A register may have additional combinational gates to perform certain operations.

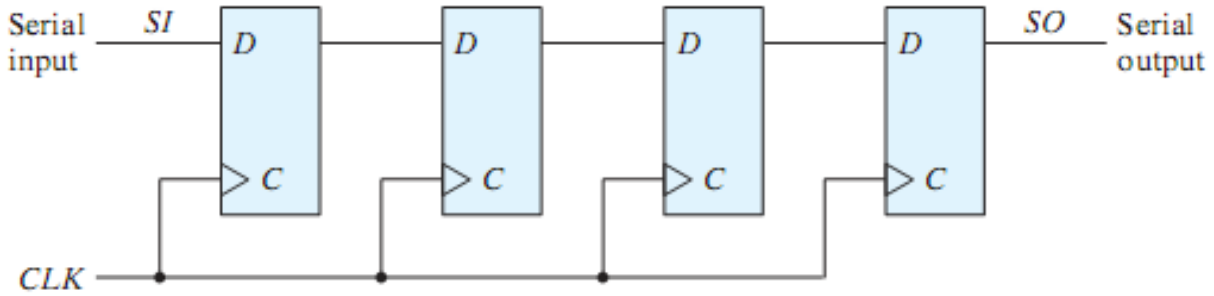
4-Bit Register

- A simple 4-bit register can be made wit
  flops.
- They have a Common Clock.
  - At each positive-edge, 4 bits are loaded in p
  - Previous data is overwritten.
- Common Clear
  - Asynchronous clear
  - When clear = 0, all flip-flops are cleared. i.e



4-bit Shift Register

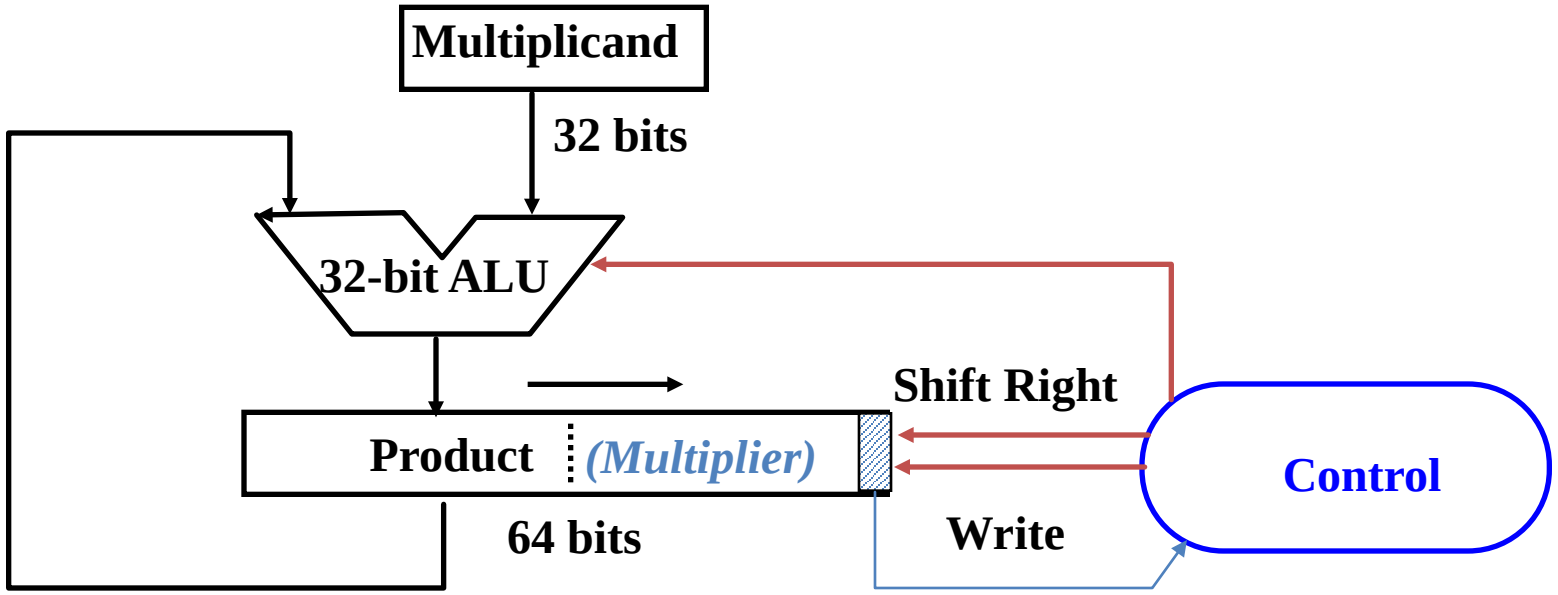
- Serial-in and Serial-out (SISO)



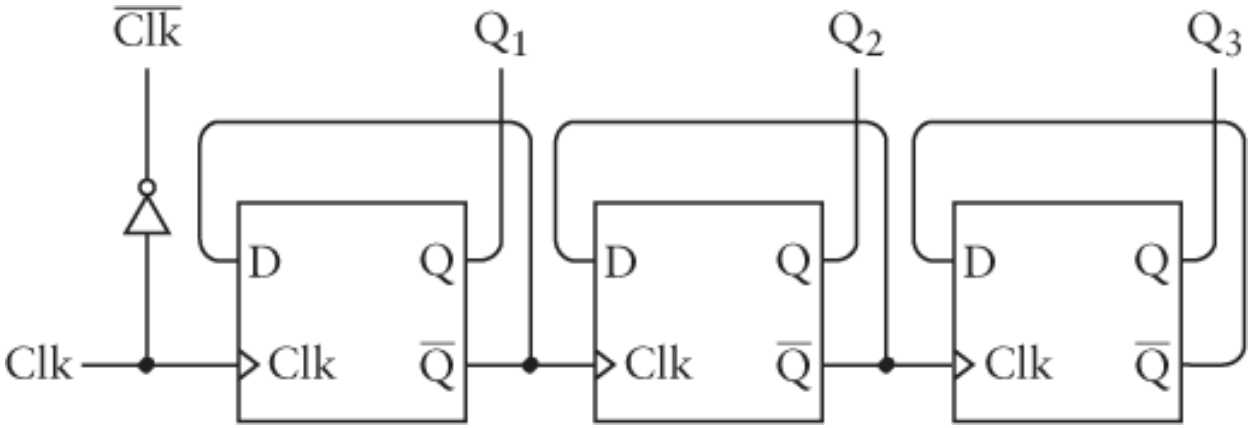
- A simple 4-bit shift register can be made with 4 D-type flip-flops.
- They have a Common Clock.
  - At each positive-edge, 1 bit is shifted in
  - Rightmost bit is discarded
- Which direction is this register shifting?

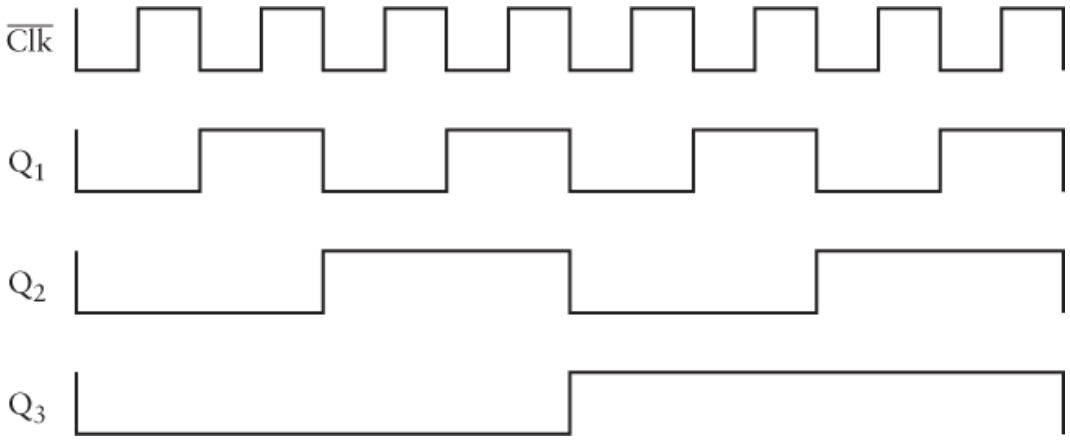
Multiply Hardware

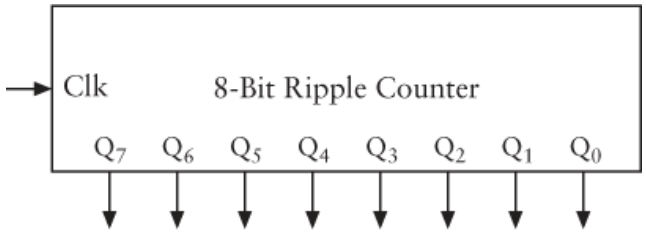
- 32-bit Multiplicand register, 32-bit ALU, 64-bit Product register, (Multiplier register is encapsulated in product register).



Frequency Divider

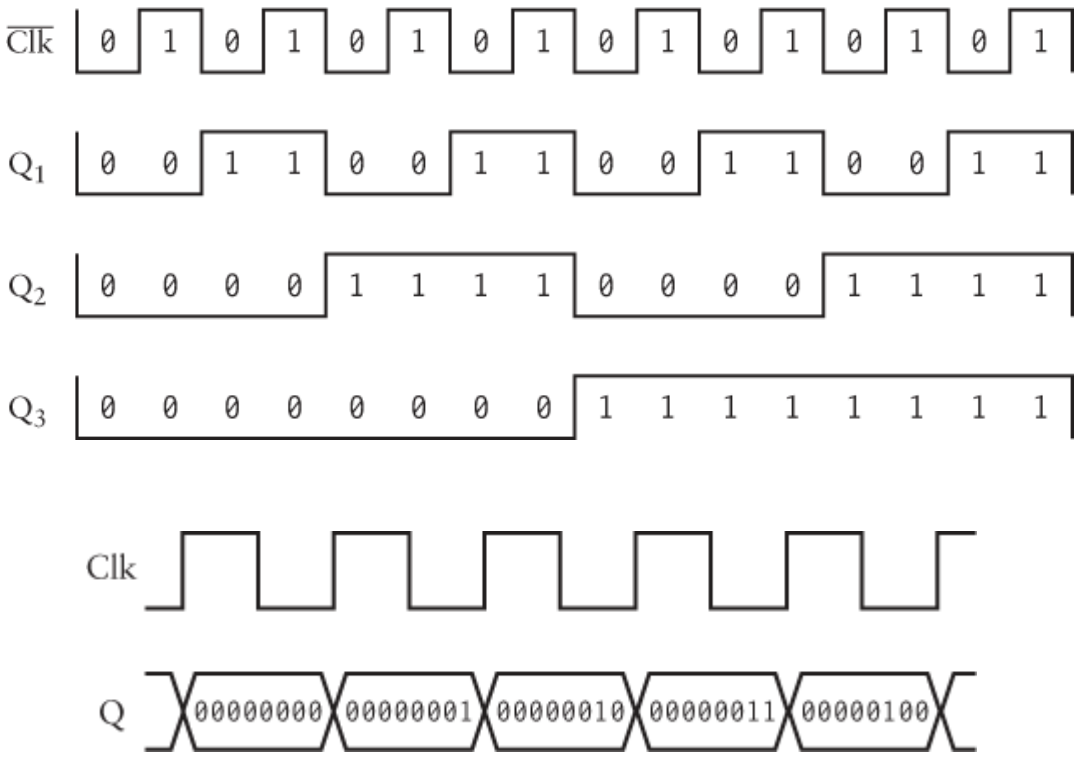


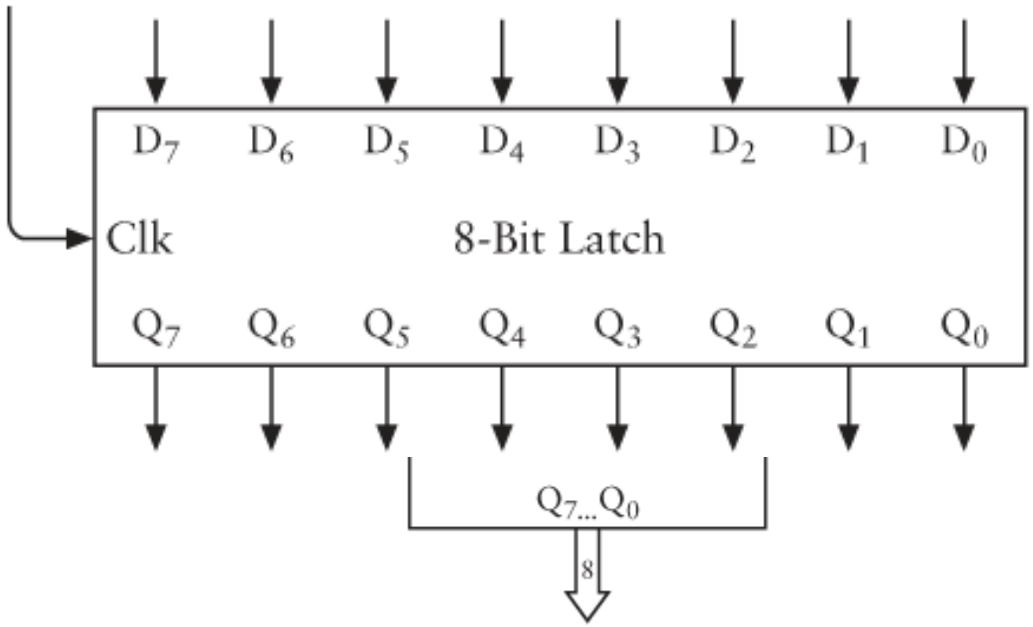




Increasing binary numbers

- At each positive transition of the Clock, some Q outputs might change and some might not, but together they reflect increasing binary numbers.




D7 D6 D5 D4 D3 D2 D1 D0
Clk
8-Bit Latch
Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0
Q7...Q0
8

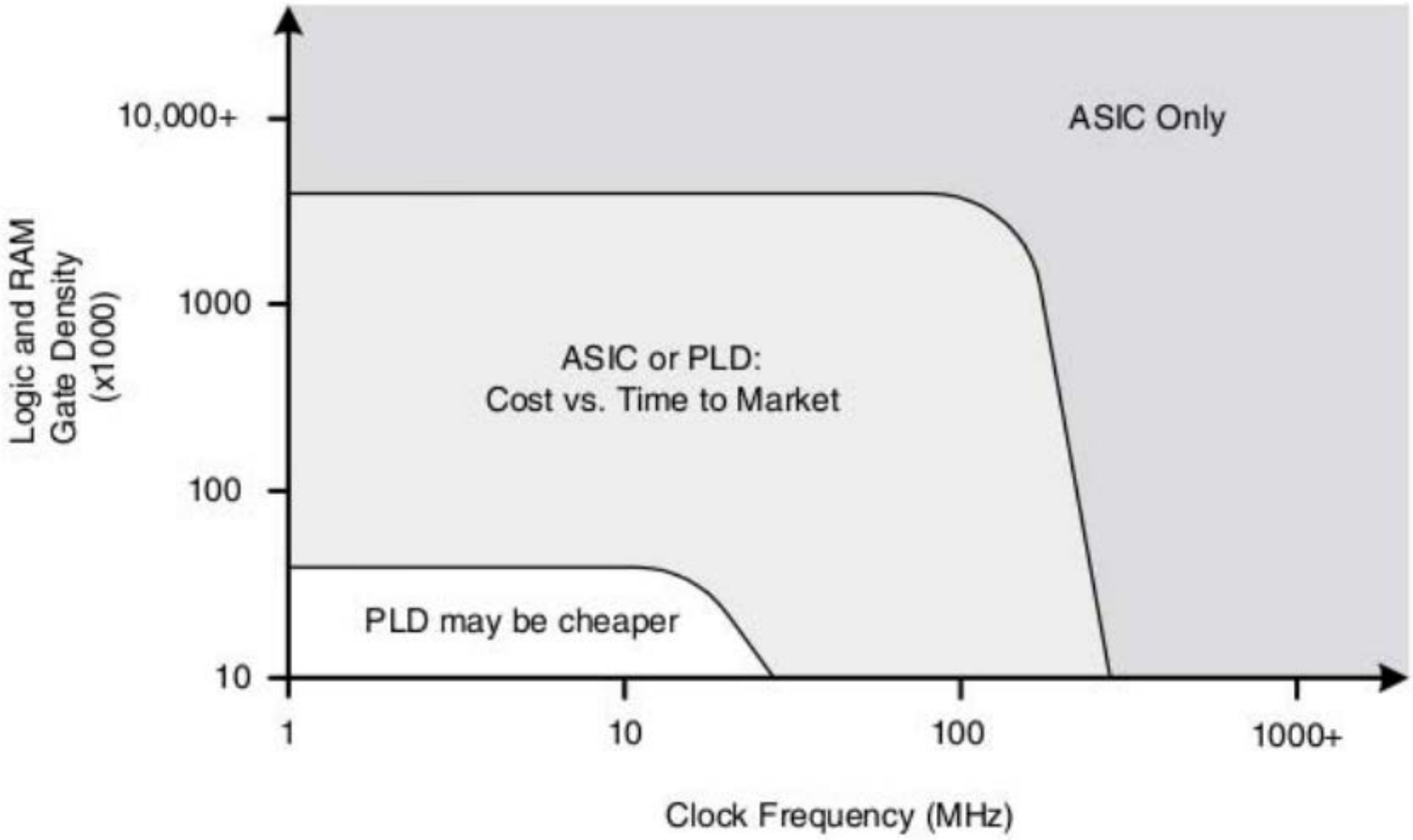
Programmable Logic Devices (PLD)

- A programmable logic device or PLD is an electronic component used to build reconfigurable digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed (i. e. reconfigured).

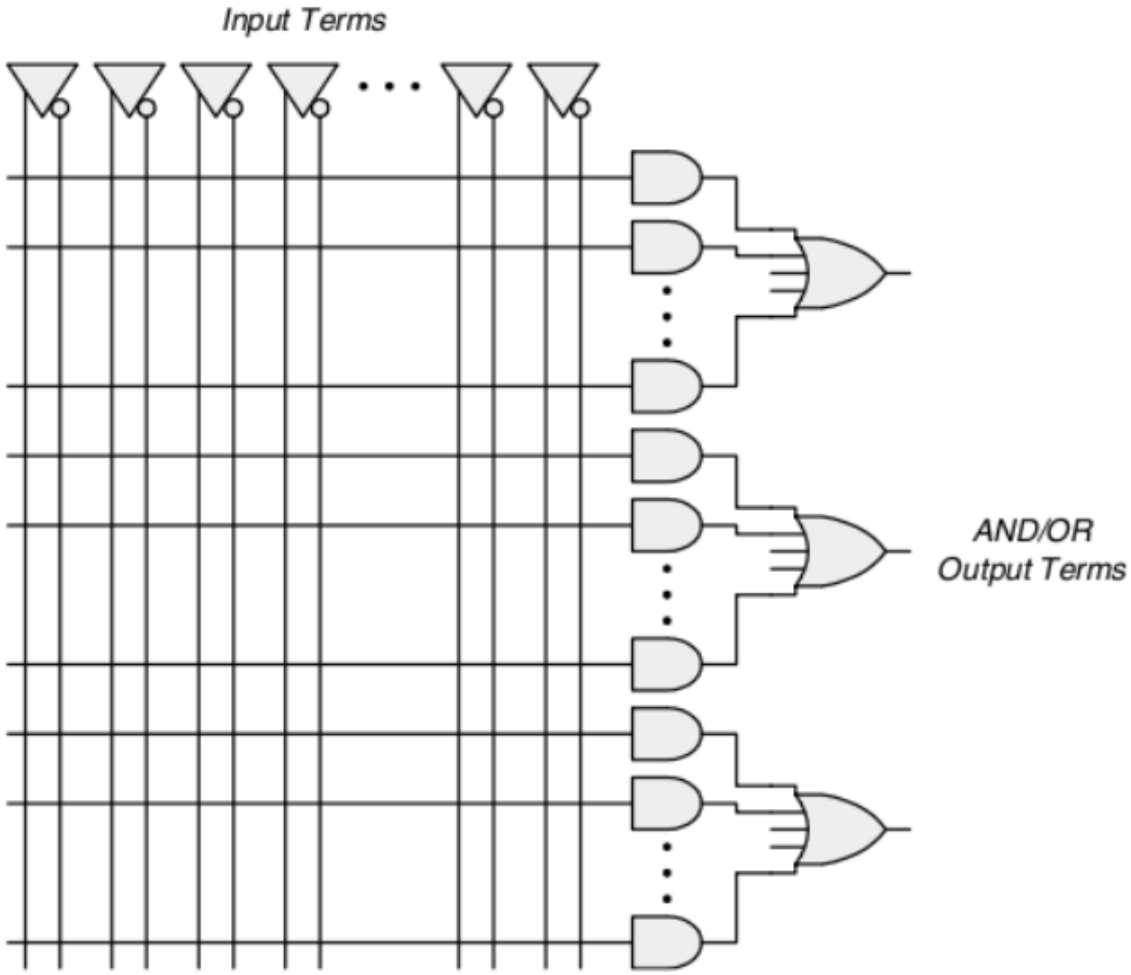
- ROM: Read Only Memory
- PAL: Programmable Array Logic
- GAL: Generic Array Logic (Derived from PAL, using OLMC—Output Logic Macro Cell)
- EPLD: Erasable Programmable Logic Device
- CPLD: Complex Programmable Logic Device
- FPGA: Field Programmable Gate Array

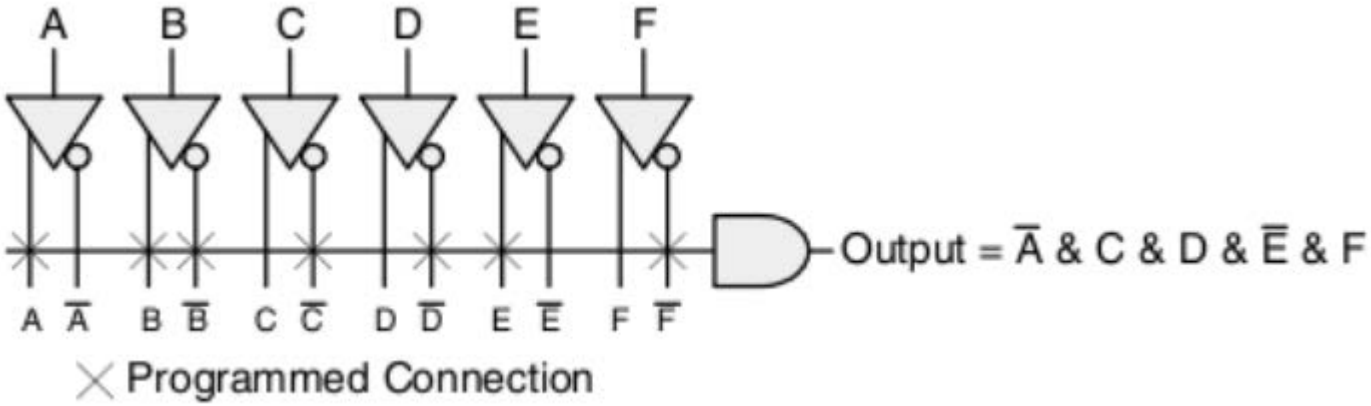
PLD vs. ASIC

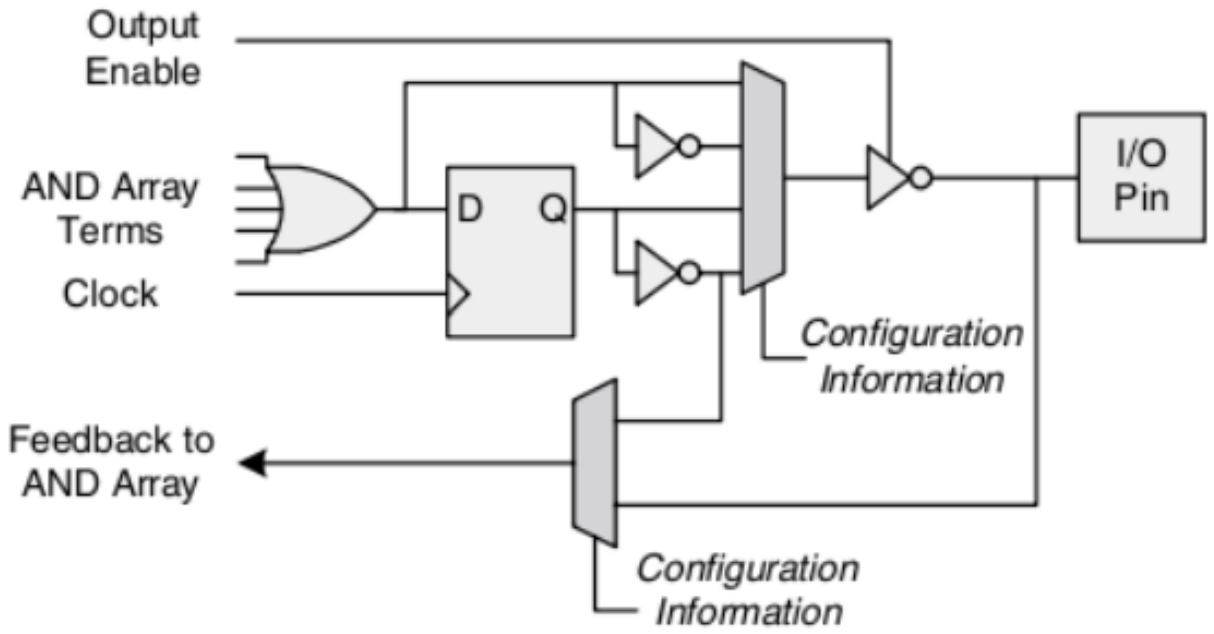
ASIC: Application Specific Integrated Circuit



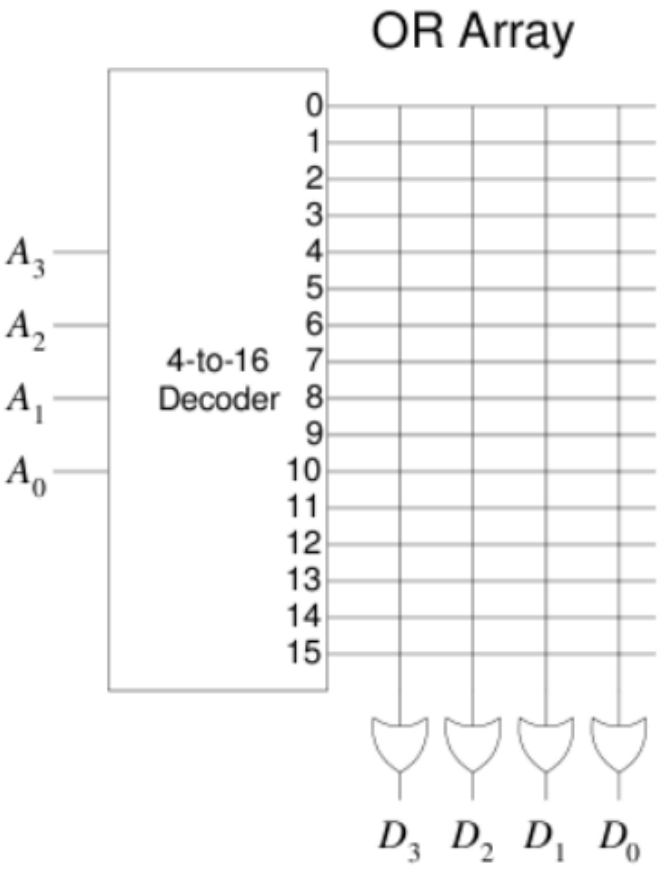
GALs and PALs

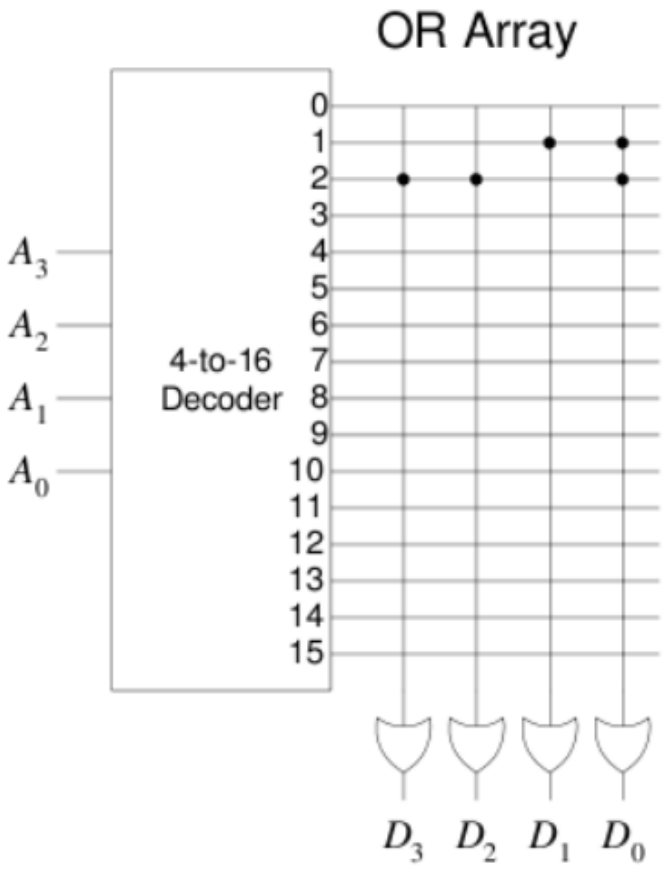



A
B
C
D
E
F
A Ā B B̄ C C̄ D D̄ E Ē F F̄
Output = Ā & C & D & Ē & F
Programmed Connection


Output Enable
AND Array Terms
Clock
D
Q
I/O Pin
Configuration Information
Feedback to AND Array
Configuration Information

Using ROMs to Implement a Function



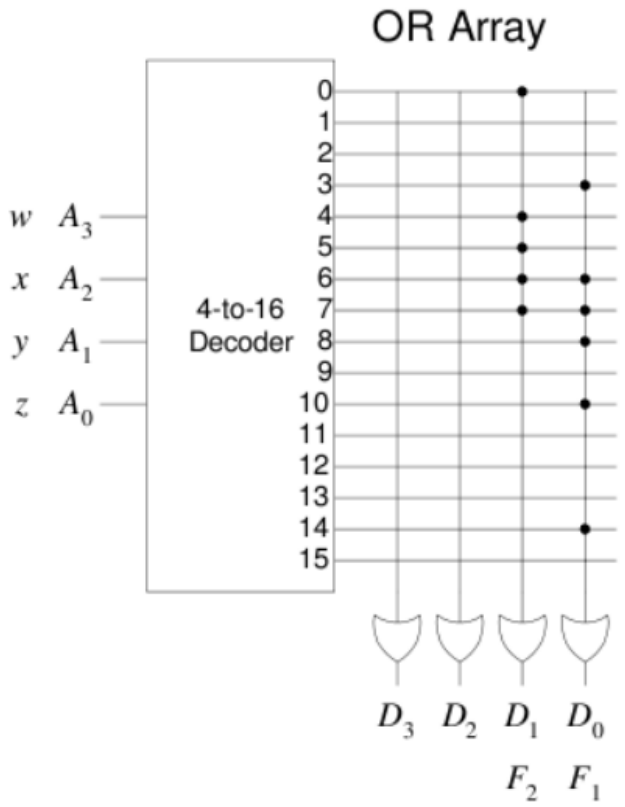


Example: Using 16x4 ROM to implement a function

- Implement the following function:

$$F_1(w,x,y,z) = w'x'yz + w'xyz' + w'xyz + wx'y'z' + wx'yz' + wxyz'$$

$$F_2(w,x,y,z) = w'x'y'z' + w'x$$



Example: Using 4x4 PAL to implement a function

Implement the following function:

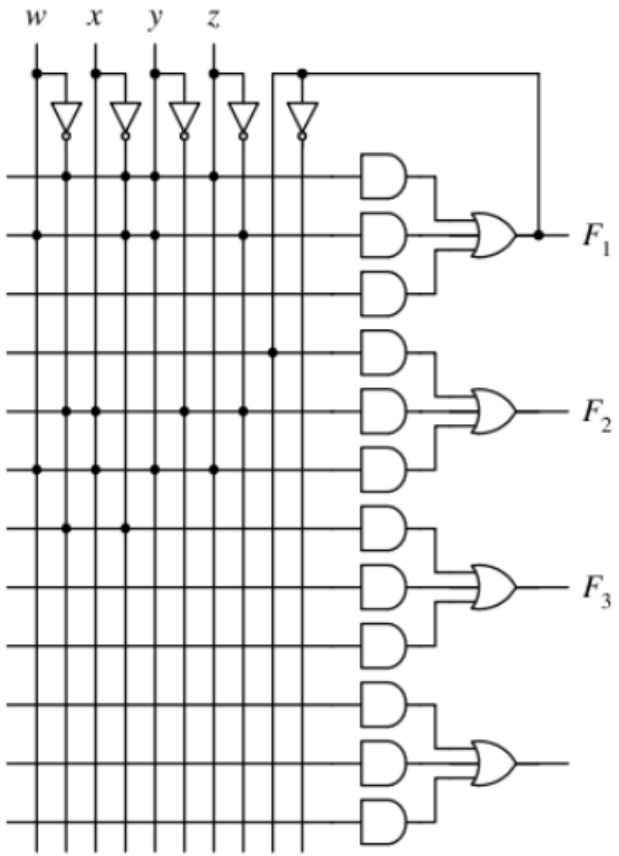
$$F_1(w,x,y,z) = w'x'yz + wx'yz'$$
$$F_2(w,x,y,z) = w'x'yz + wx'yz' + w'xy'z' + wxyz$$
$$F_3(w,x,y,z) = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz$$

First we can reduce the terms like as follows:

$$F_2(w,x,y,z) = w'x'yz + wx'yz' + w'xy'z' + wxyz$$
$$= F_1 + w'xy'z' + wxyz$$

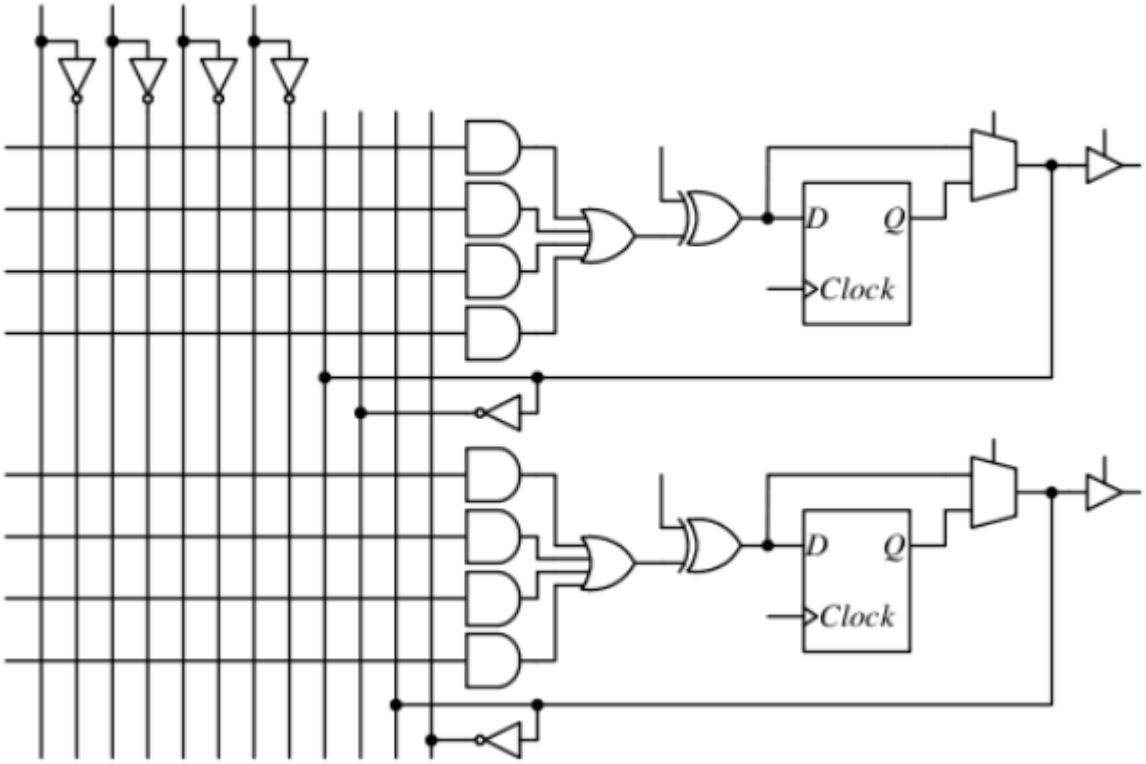
$$F_3(w,x,y,z) = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz$$
$$= w'x'(y'z' + y'z + yz' + yz)$$
$$= w'x'$$

=>

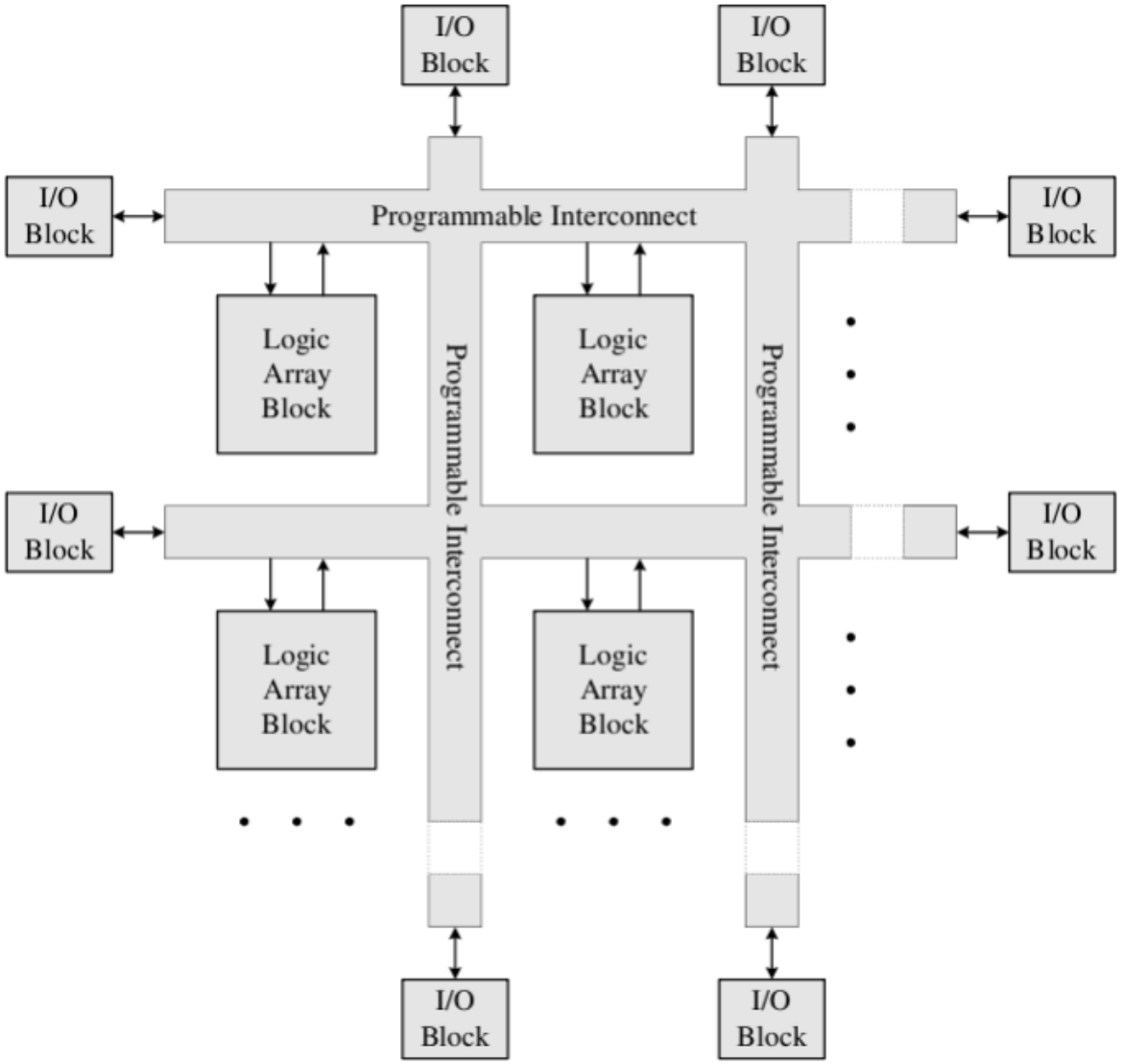


Complex Programmable Logic Device (CPLD)

- (CPLD) is capable of implementing a circuit with upwards of 10,000 logic gates. Sequential circuits can also be implemented with CPLDs.

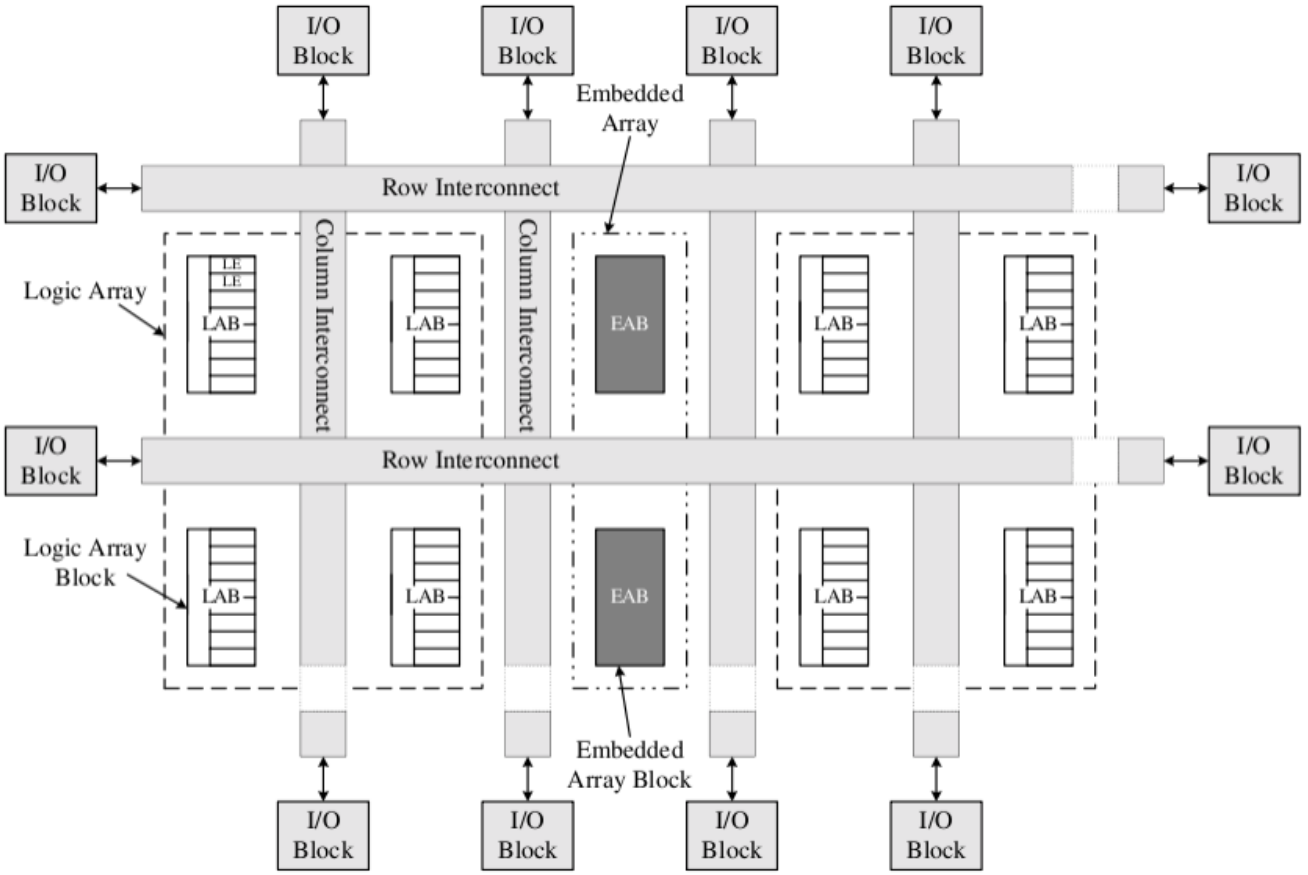


Internal Structure of CPLDs

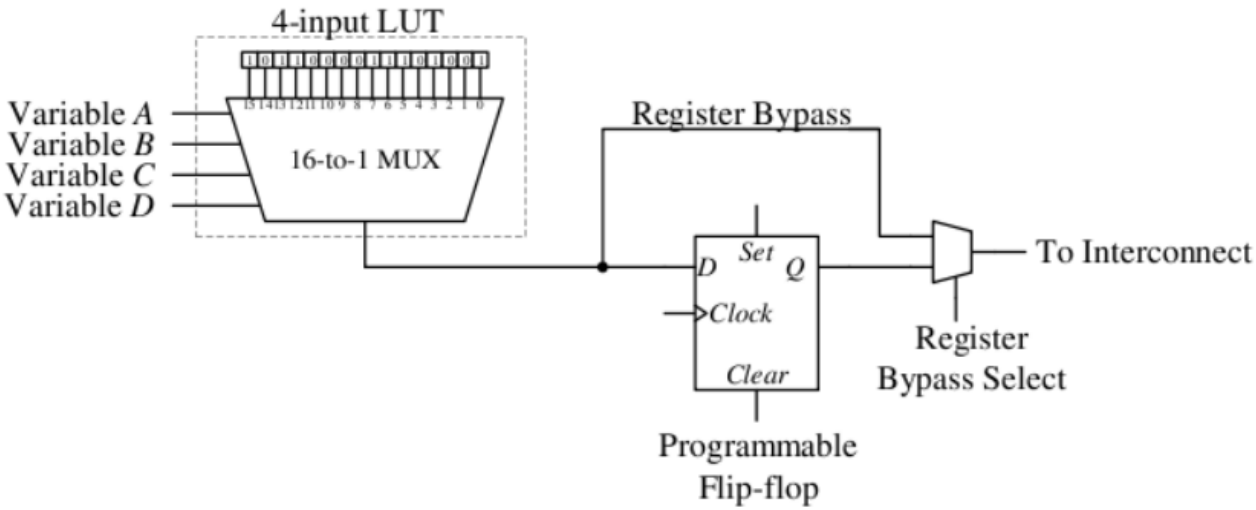


Field Programmable Gate Array (FPGA)

- Field programmable gate arrays (FPGAs) are complex programmab
  implementin
  RAM bits, as
  chip.



FPGA Internal Structure



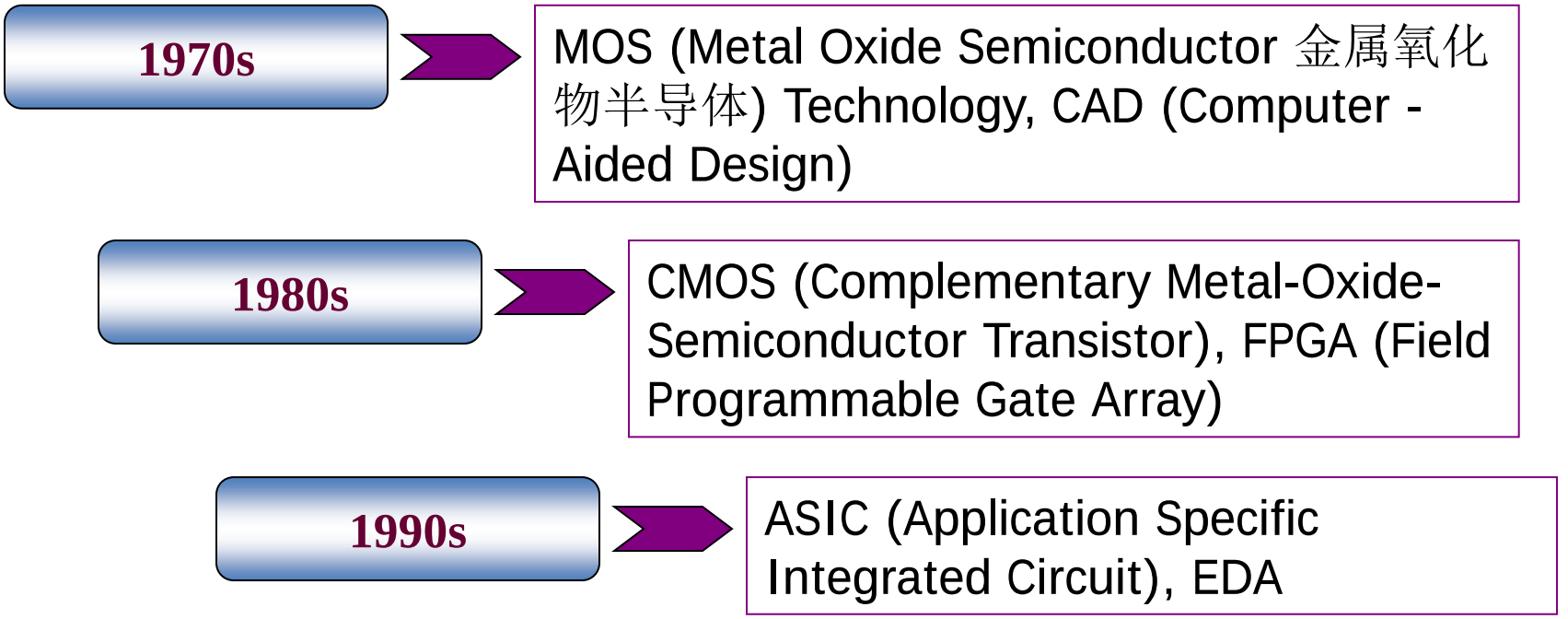
All the EABs, LABs, and I/O elements, are connected together via the FastTrack interconnect, which is a series of fast row and column buses that run the entire length and width of the device. The interconnect contains programmable switches so that the output of any block can be connected to the input of any other block.

Each I/O pin in an I/O element is connected to the end of each row and column of the interconnect and can be used as either an input, output, or bi-directional port.

EDA: Electronic Design Automation

The core of modern electronic design technology has increasingly turned to electronic design automation technology based on computer

Three stages of the development of EDA technology



Programming Language

- Hardware Description Language (HDL)
  - Can we use C or Java as HDL?
- A computer programming language
  - Semantics (“meaning”)
  - Syntax (“grammar”)
- Develop of a language
  - Study the characteristics of the underlying processes.
  - Develop syntactic constructs and their associated semantics to model and express these characteristics.

Traditional PL

- Modeled after a sequential process
  - Operations performed in a sequential order
  - Help human's thinking process to develop an algorithm step by step
  - Resemble the operation of a basic computer model

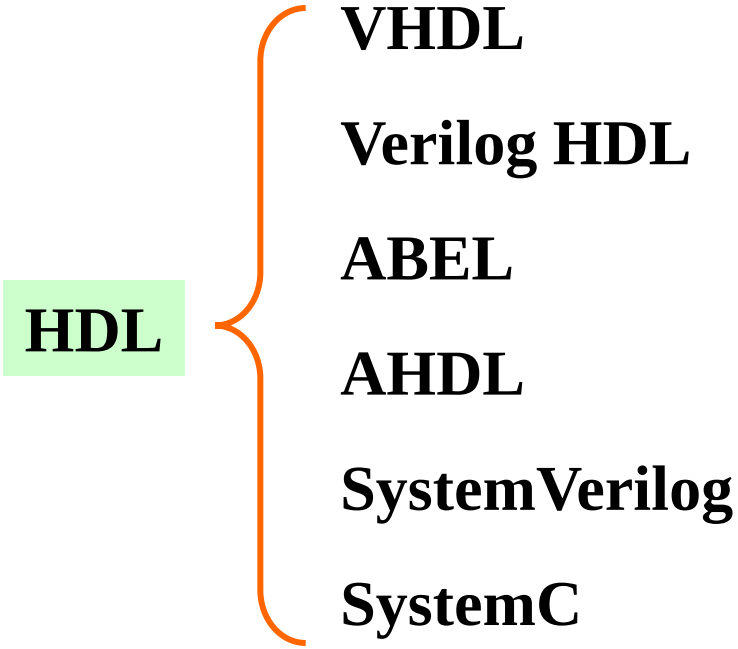
HDL

- Characteristics of digital hardware
  - Connections of parts
  - Concurrent operations
  - Concept of propagation delay and timing
- Characteristics cannot be captured by traditional PLs
- Require new languages: HDL

Modern HDL

- Capture characteristics of a digital circuit:
  - entity
  - connectivity
  - concurrency
  - timing
- Cover description
  - In Gate level and Register-Transfer level (RTL)
  - In structural view and behavioral view

HDL



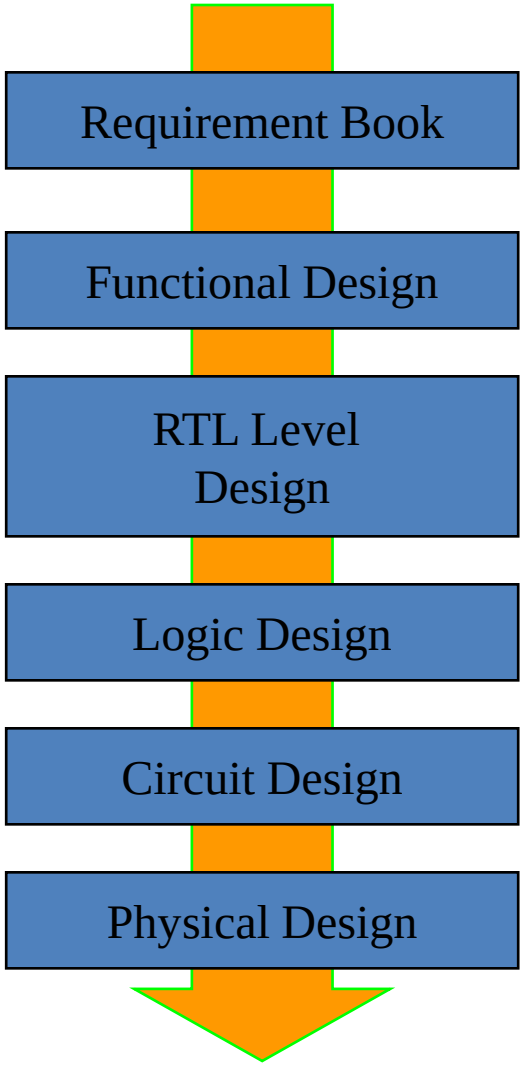
History of VHDL

- VHDL
  - VHSIC (Very High Speed Integrated Circuit)
  - Hardware
  - Description
  - Language
- 1980——VHSIC plan of the US department of defense
- 1985——First Version
- 1987——IEEE Standard 1076 (VHDL’87)
- 1993——IEEE Standard 1076 (VHDL’93)

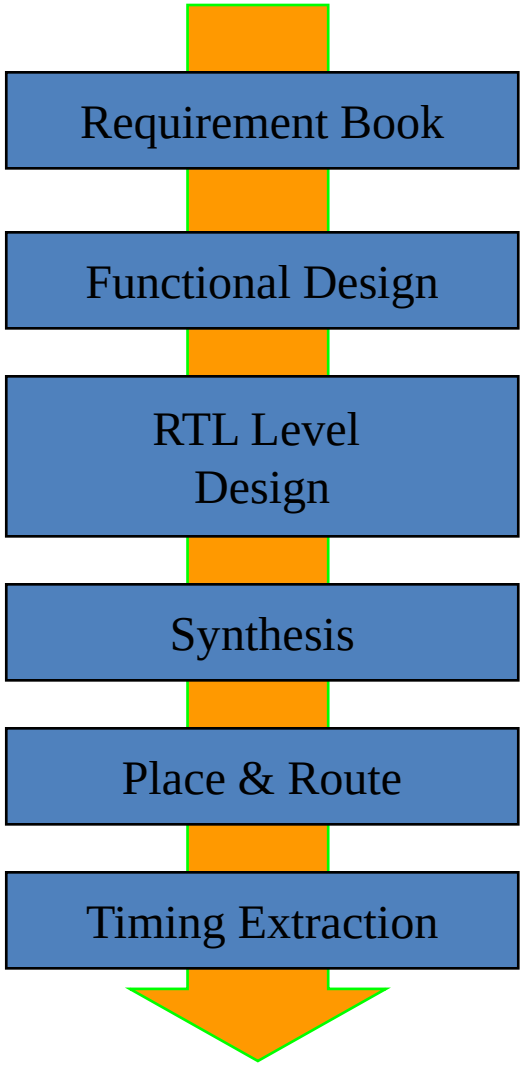
Why using VHDL?

- A Language For Synthesis
  - Independent to its physical implementation
  - A standard accepted by most EDA (synthesis) software
  - Easy to maintain
  - Code reuse
- A Language For Simulation
  - Supported by most EDA (simulator) tools
  - Help engineers verify their design
  - HW-SW mixed system design

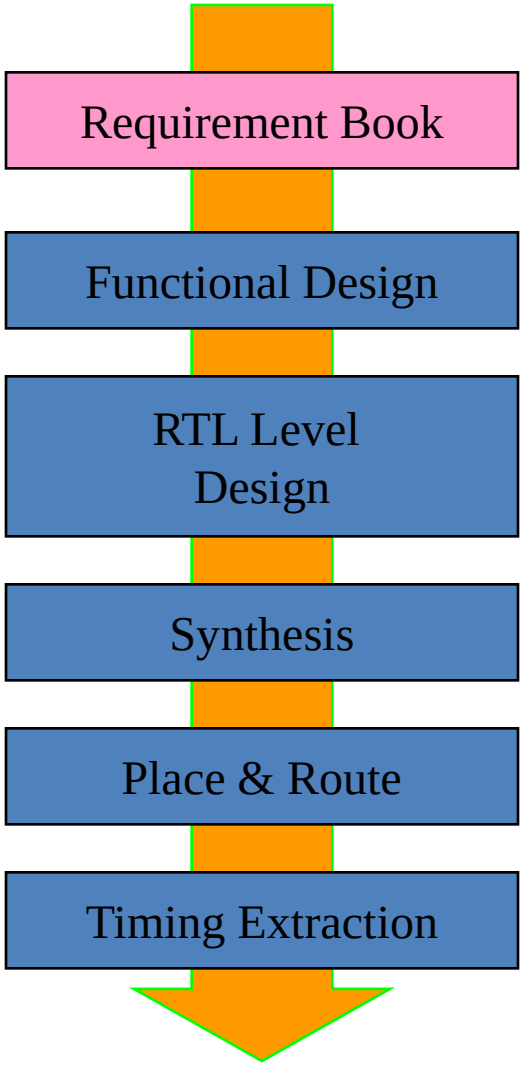
Digital System Design Flowchart (ASIC)



Digital System Design Flowchart (FPGA)

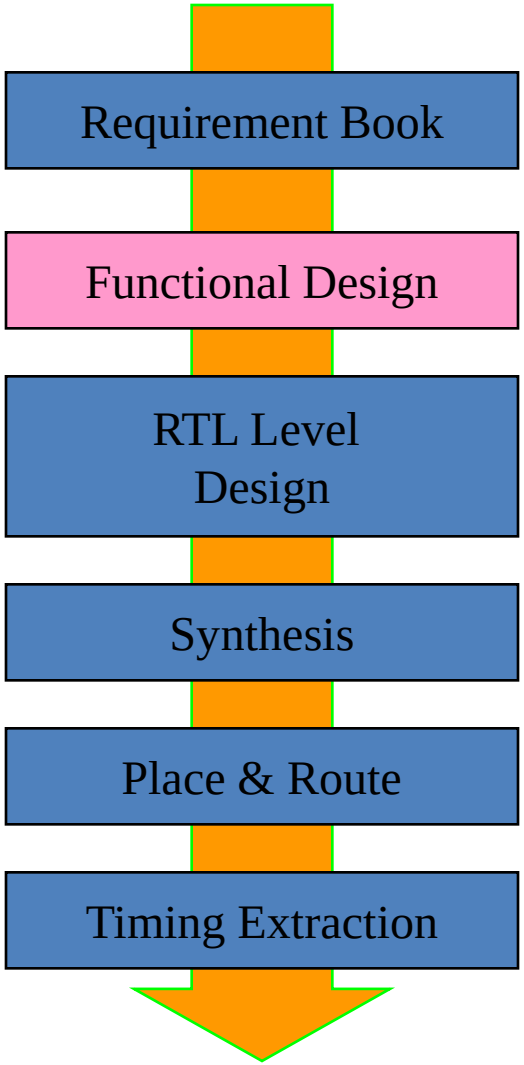


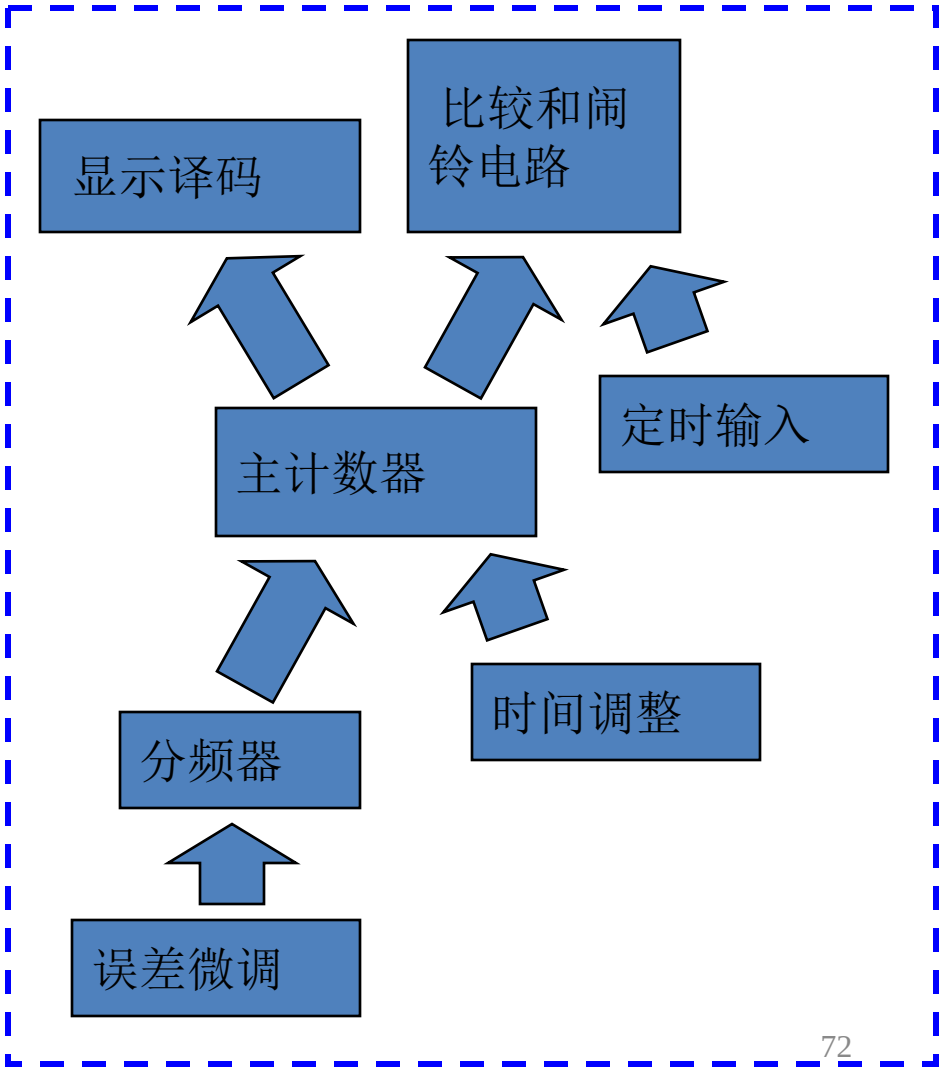
Flowchart Example (1)



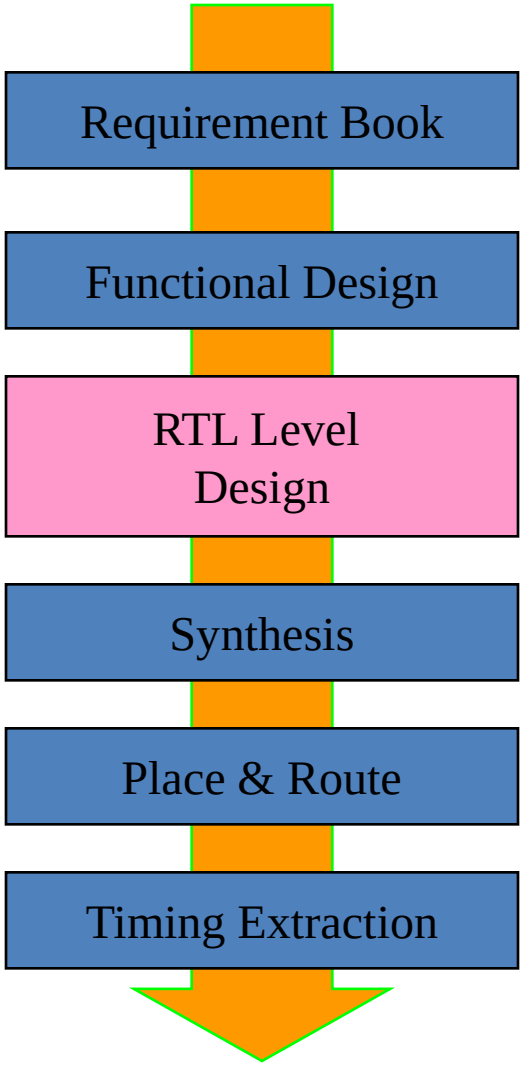
- 电子钟
- 定时闹铃
- 手动调时
- 定时误差微调

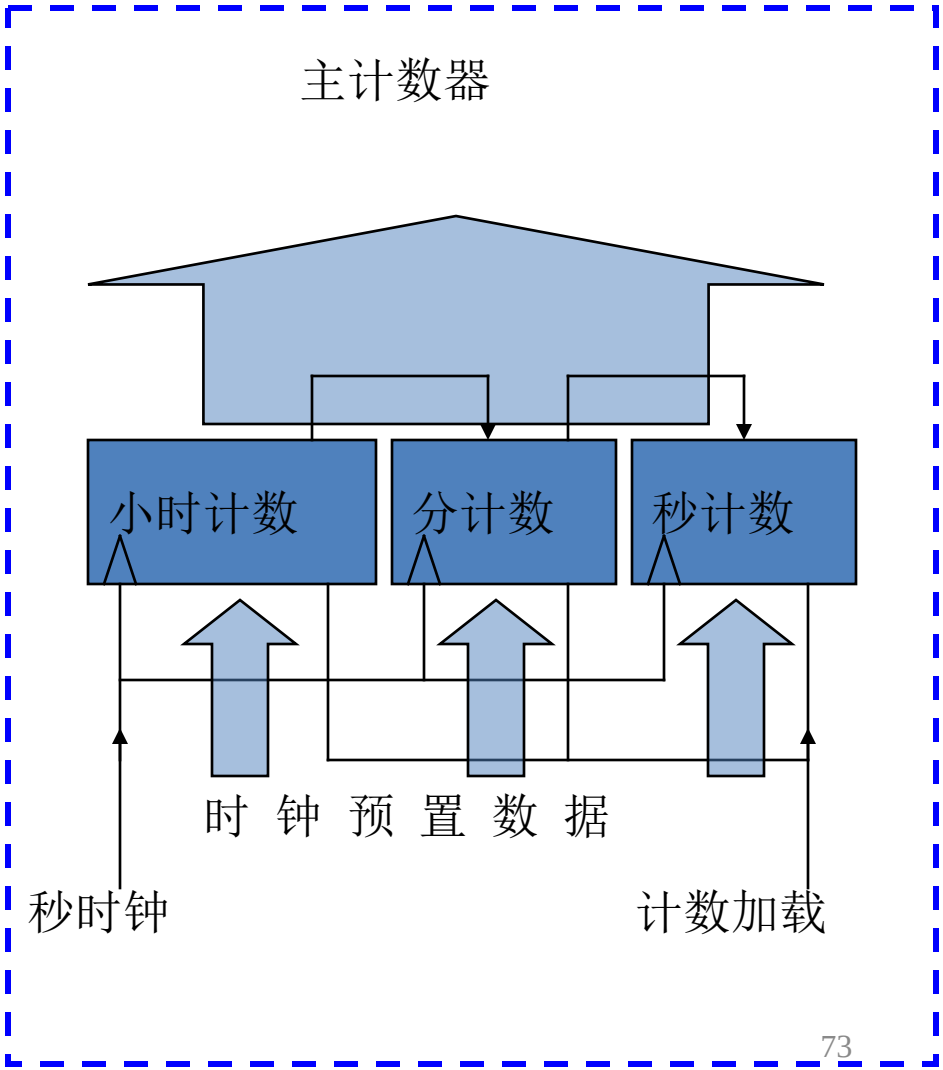
Flowchart Example (2)



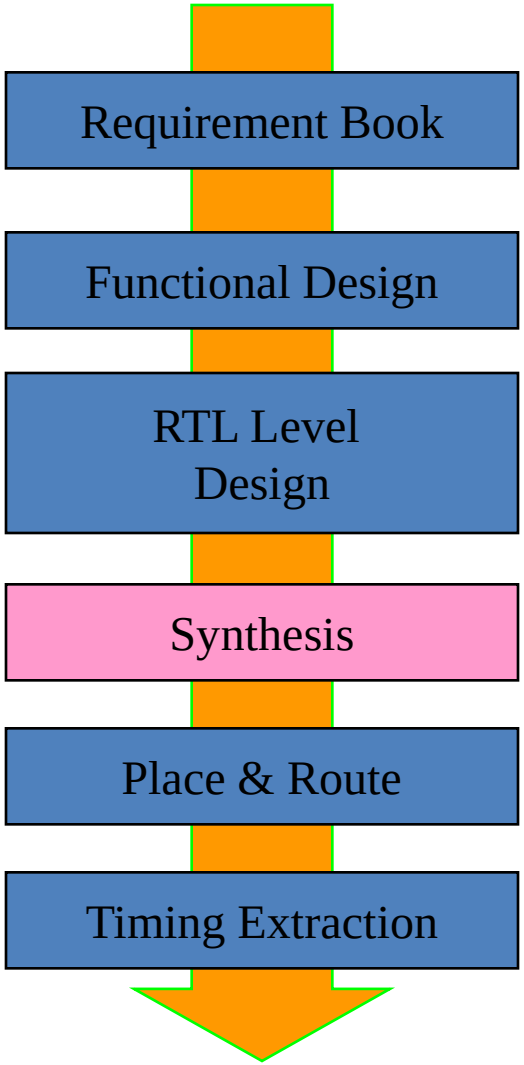


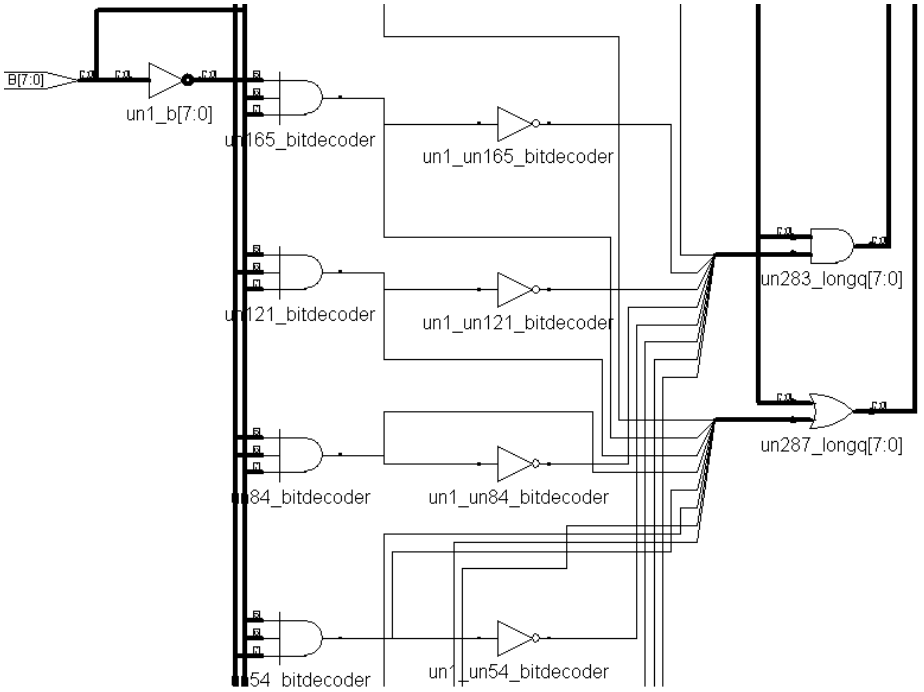
Flowchart Example (3)



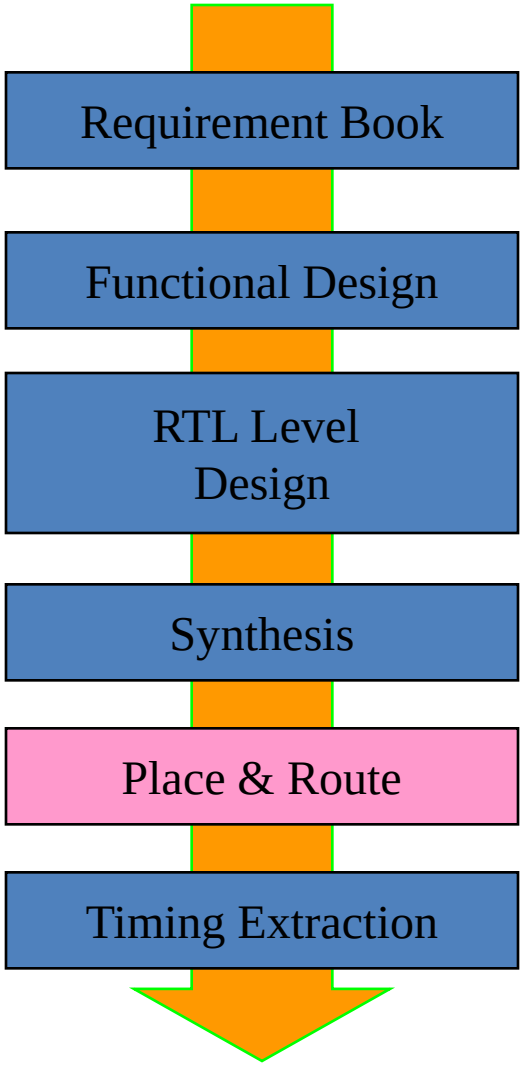


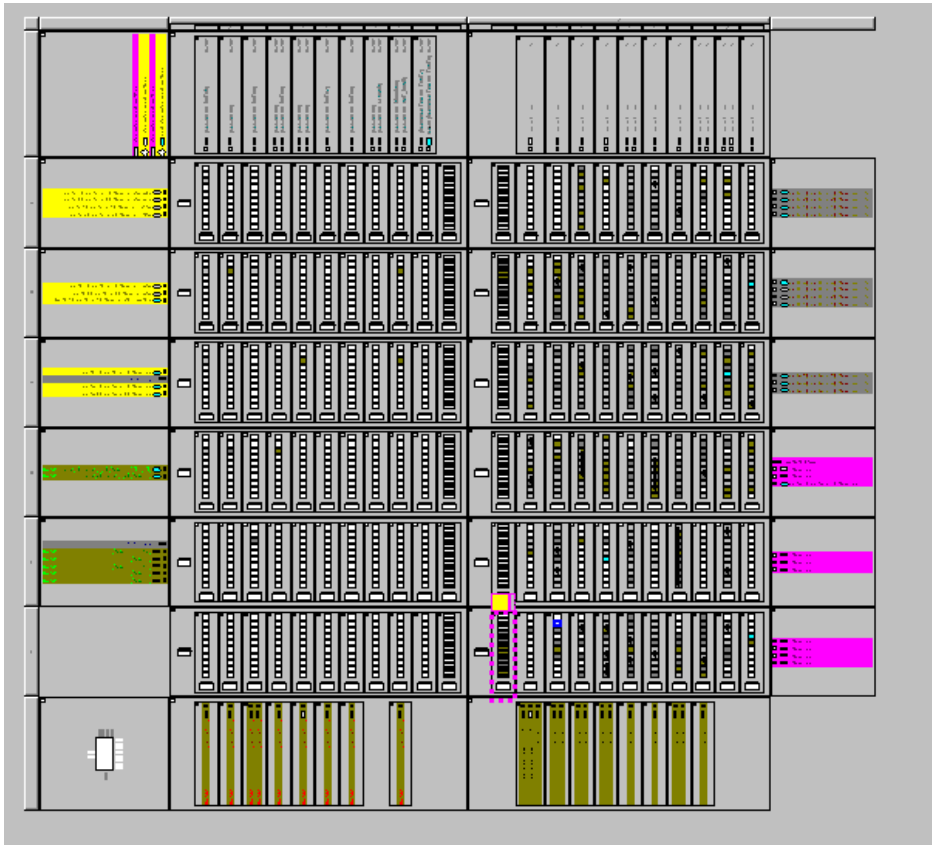
Flowchart Example (4)



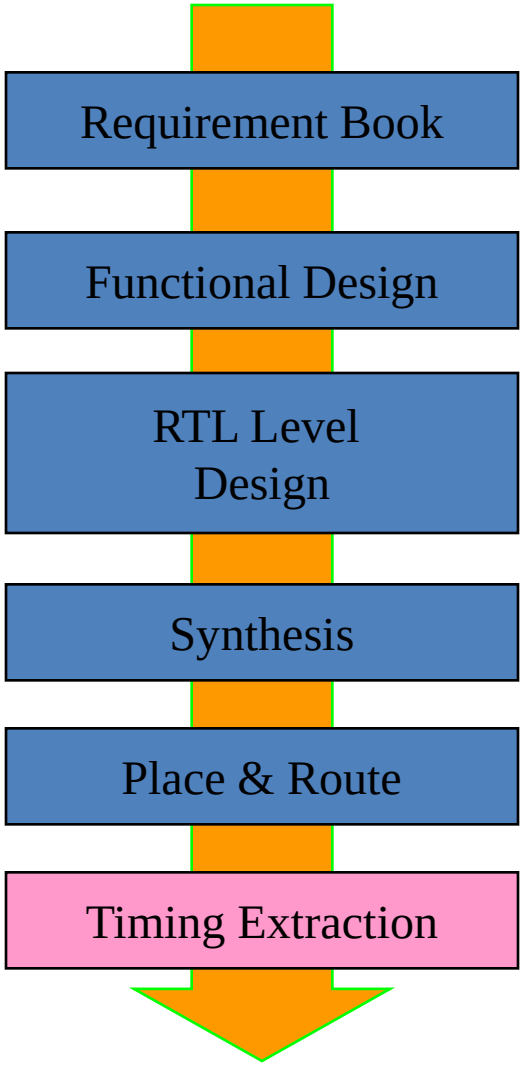


Flowchart Example (5)





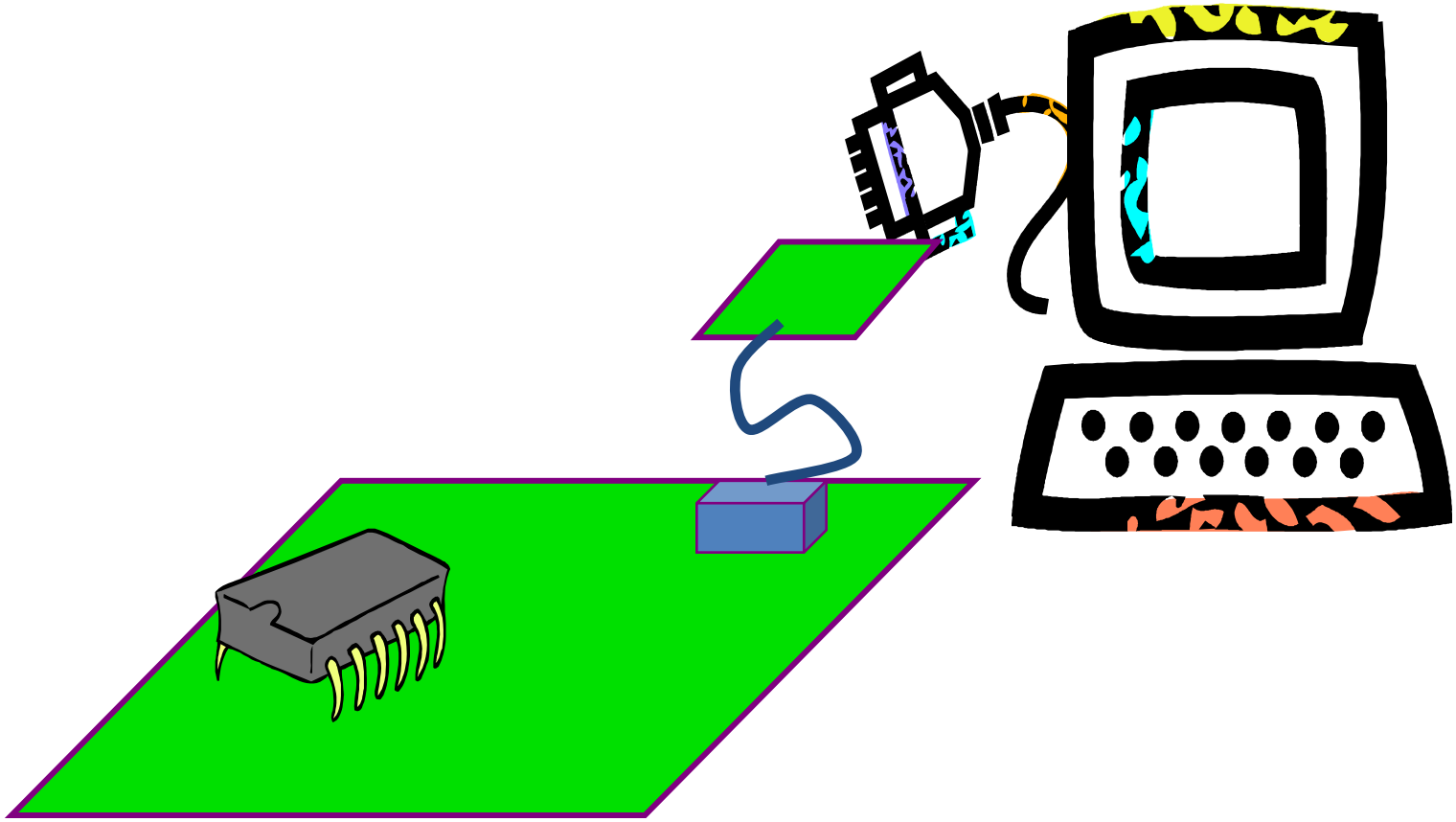
Flowchart Example (6)



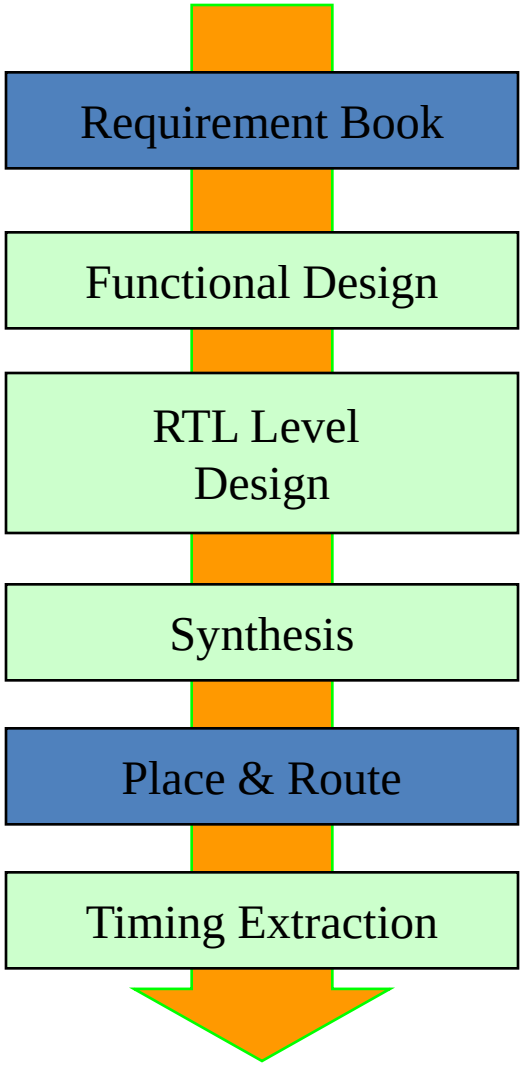
Timing Analyzer Summary

	Type	Slack	Required...	Actual Time	Source Name	Destination Name
1	Clock Setup: 'clock'	985.989 ns	1.00 MHz (...	71.37 MHz (perio...	INS_Decode:D...	ALU:UC_ALU|ALUZ
2	Worst-case tsu	N/A	None	11.489 ns	portc[6]	Data_Mux:Dmux|A...
3	Worst-case tco	N/A	None	10.414 ns	Spcl_Regs:SP...	portc[5]
4	Worst-case th	N/A	None	-4.776 ns	portb[3]	Data_Mux:Dmux|A...
5	Worst-case minimum tco	N/A	None	6.035 ns	Spcl_Regs:SP...	portb[1]

Hardware Implementation



Where is the VHDL?



Behavior model in VHDL for function simulation

RTL model in VHDL for logic simulation & synthesis

VHDL code → EDA Tools → physical implementation

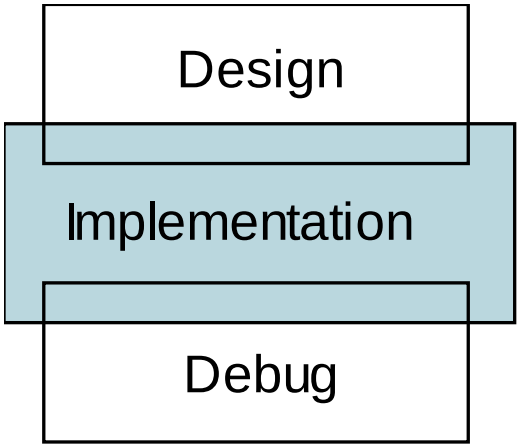
Simulation

Conclusion

- VHDL is Good

- Describe hardware in different level in digital system design
- Accepted by most EDA tools to help implement digital design
- Help simulating the digital product at different design stage
- Concurrent execution

The Process of Design



Design

Initial concept: what is the function performed by the object?
Constraints: How fast? How much area? How much cost?
Refine abstract functional blocks into more concrete realizations

Implementation

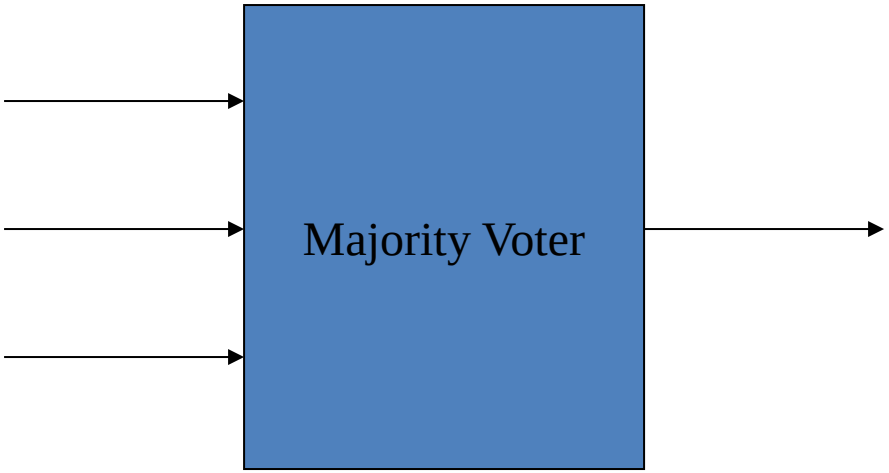
Assemble primitives into more complex building blocks
Composition via wiring
Choose among alternatives to improve the design

Debug

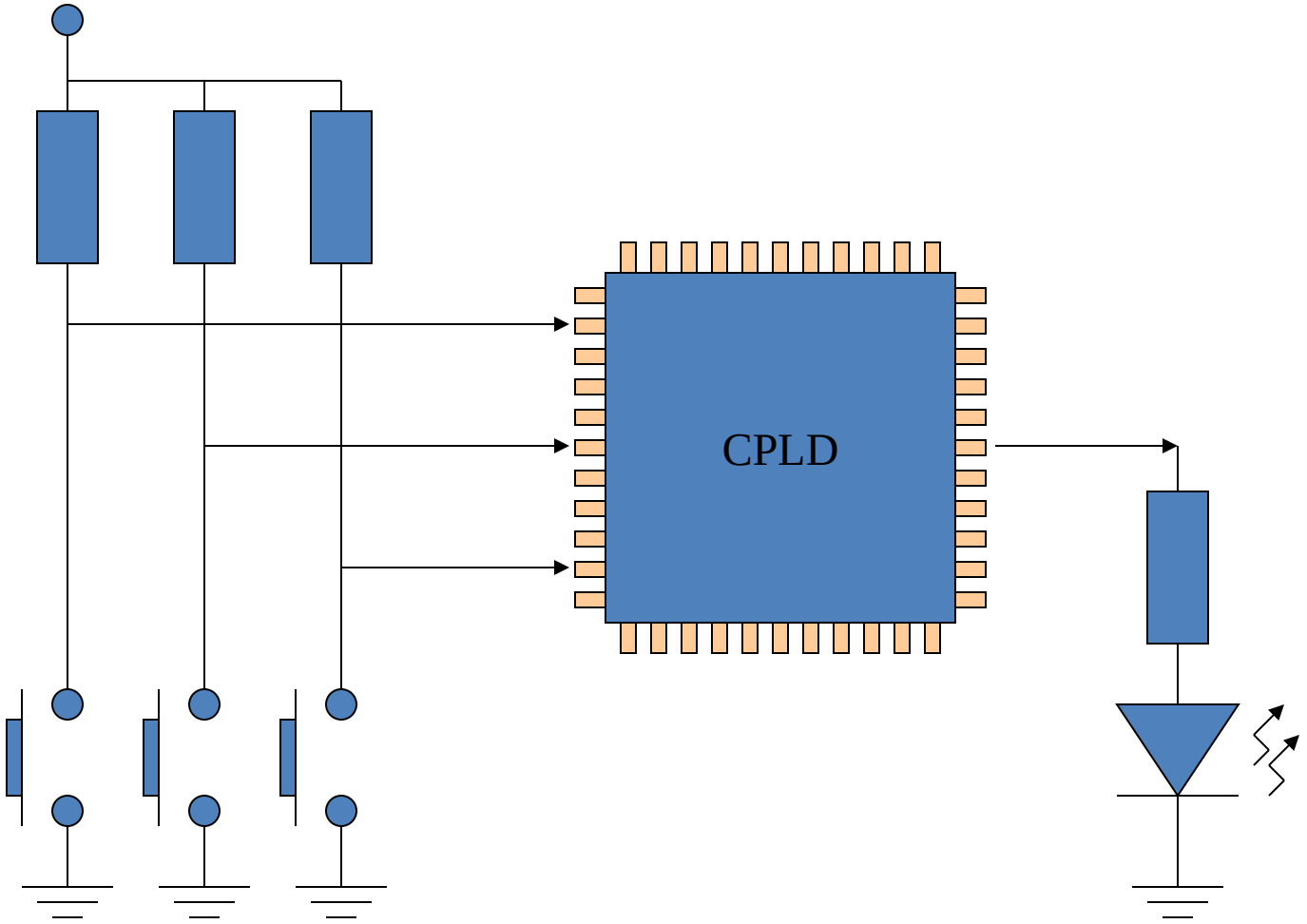
Faulty systems: design flaws, composition flaws, component flaws
Design to make debugging easier
Hypothesis formation and troubleshooting skills

Majority Voter Circuit

- Output ‘1’ of there is more ‘1’s than ‘0’s
- Output ‘0’ of there is more ‘0’s than ‘1’s



Schematic

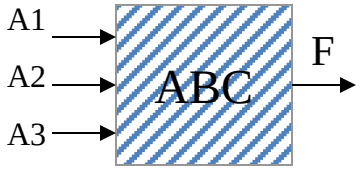


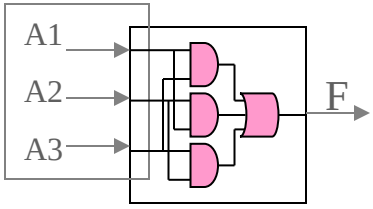
VHDL Description

File

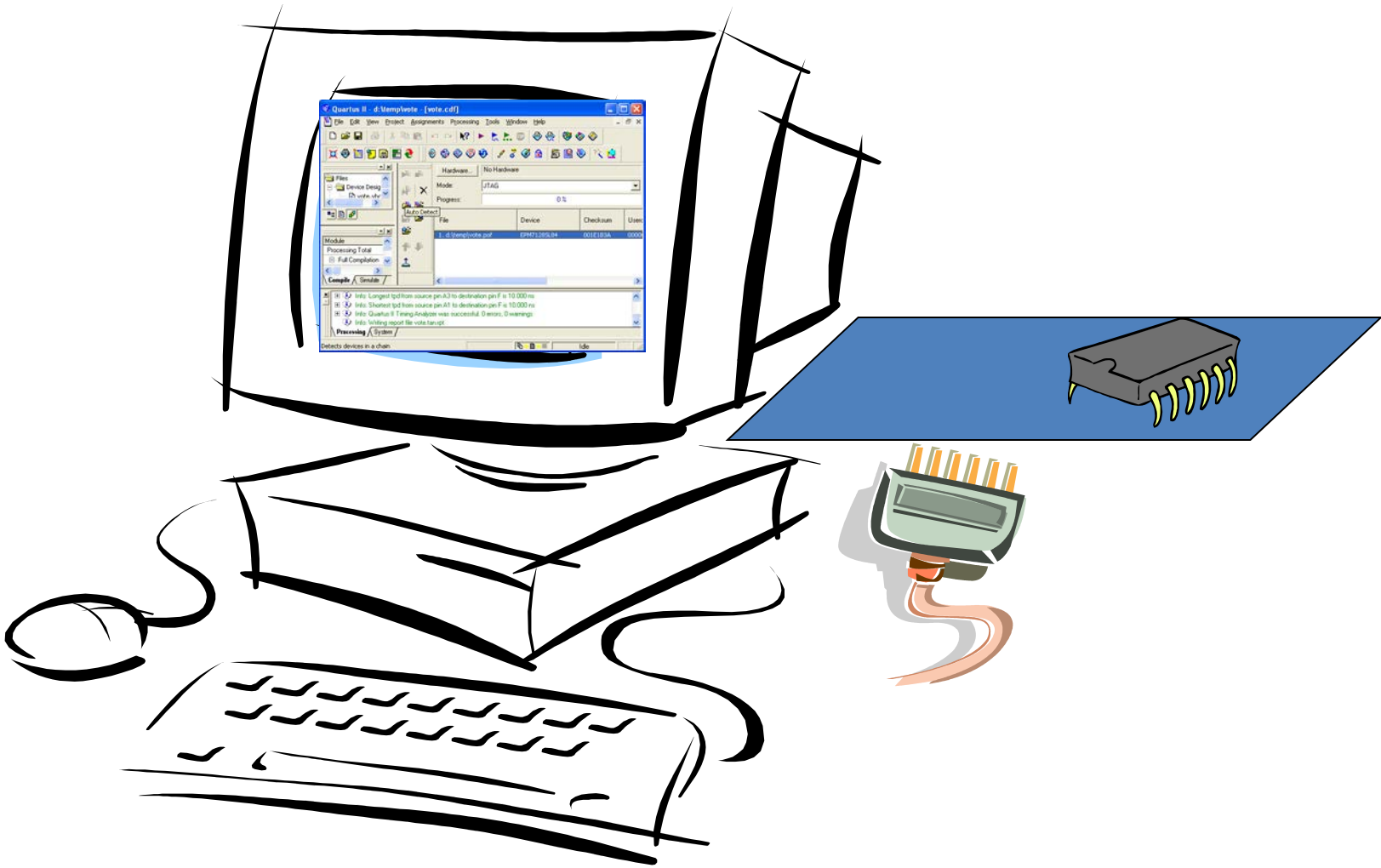
XYZ.VHD

```
library ieee;
use ieee.std_logic_1164.all;
--------------------------------
entity XYZ
port
   (  A1, A2, A3  :  in  std_logic;
      F           :  out std_logic );
end XYZ;
--------------------------------
architecture XYZ _arch of XYZ is
begin
   F <= (A1 and A2) or (A2 and A3) or (A1 and A3);
end XYZ_arch;
```

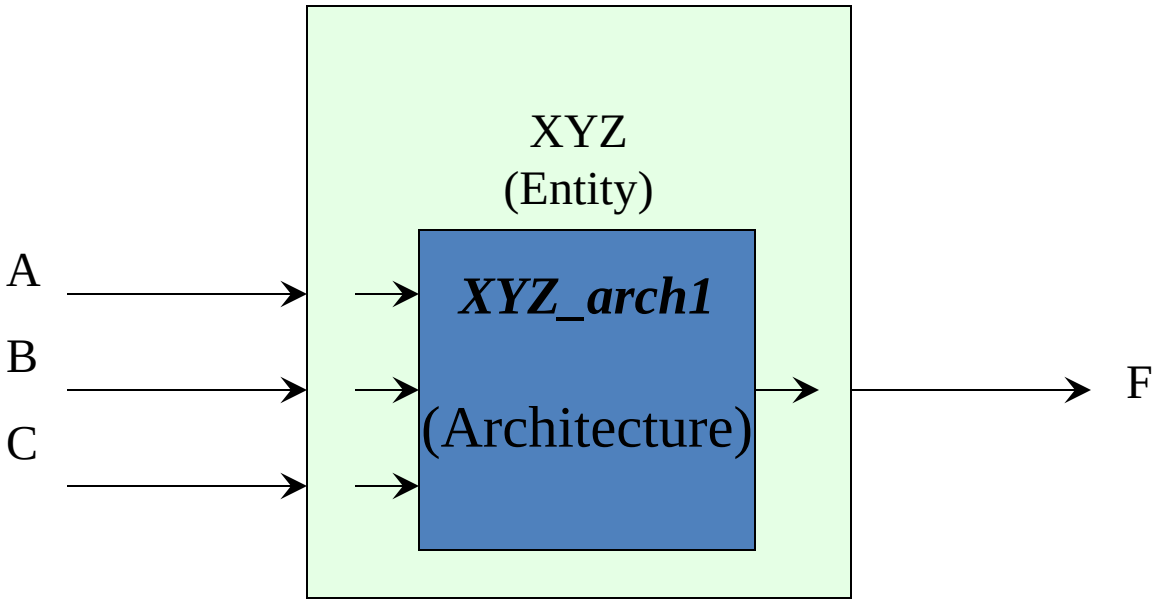




Download to Chips



XYZ Clock Diagram



XYZ.VHD (1) data flow

```
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------
entity XYZ is
    port
    (
        A, B, C      :   in  std_logic; -- Comments
        F            :   out std_logic
    );
end XYZ;
----------------------------------------
architecture XYZ_arch of XYZ is
begin

    F <= (A and B) or (B and C) or (C and A);

end XYZ_arch;
```

XYZ.VHD (2) data flow

```
library ieee;
use ieee.std_logic_1164.all;
-------------------------------
entity XYZ is
    port
    (
        A, B, C        :   in  std_logic;
        F              :   out std_logic
    );
end XYZ;
-------------------------------
architecture XYZ_arch of XYZ is
begin
    F <= '1' when ((A = '1') and (B = '1')) else
         '1' when ((B = '1') and (C = '1')) else
         '1' when ((A = '1') and (C = '1')) else
         '0';
end XYZ_arch;
```

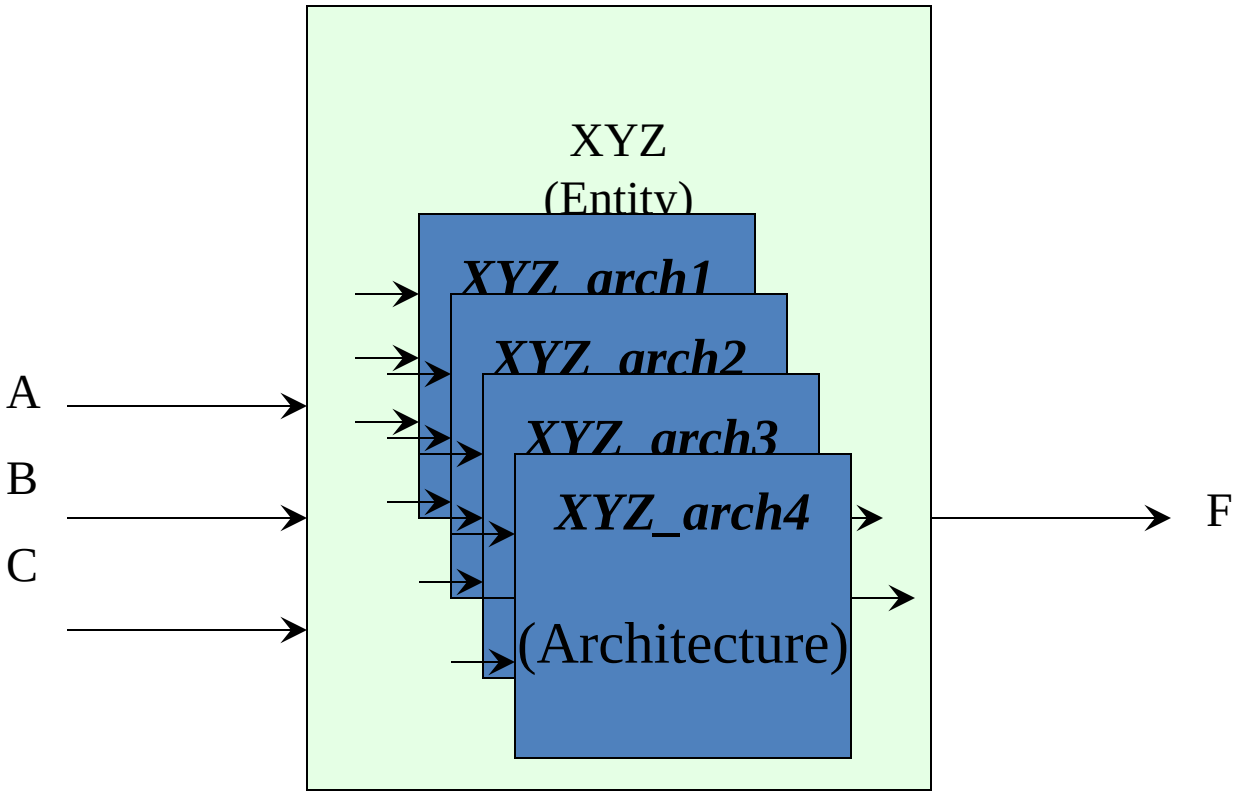
XYZ.VHD (3) behavioral

```
library ieee;
use ieee.std_logic_1164.all;
--------------------------------
entity XYZ is
    port
    (
        A, B, C        :   in  std_logic;
        F              :   out std_logic
    );
end XYZ;
--------------------------------
architecture XYZ_arch of XYZ is
begin
    process (A, B, C)
    begin
        if (A = '1') and (B = '1') then
            F <= '1';
        elsif (B = '1') and (C = '1') then
            F <= '1';
        elsif (A = '1') and (C = '1') then
            F <= '1';
        else
            F <= '0';
        end if;
    end process;
end XYZ_arch;
```

XYZ.VHD (4) structural

```
library ieee;
use ieee.std_logic_1164.all;
library altera;
use altera.maxplus2.ALL;
---------------------------------
entity XYZ is
    port
    (
        A, B, C          :   in  std_logic;
        F                :   out std_logic
    );
end XYZ;
---------------------------------
architecture XYZ_arch of XYZ is
    signal F1, F2, F3: std_logic;
begin
    U1: and2
        port map(A, C, F1);
    U2: and2
        port map(A, B, F2);
    U3: and2
        port map(B, C, F3);
    F <= F1 or F2 or F3;
end XYZ_arch;
```

XYZ All in One Block Diagram



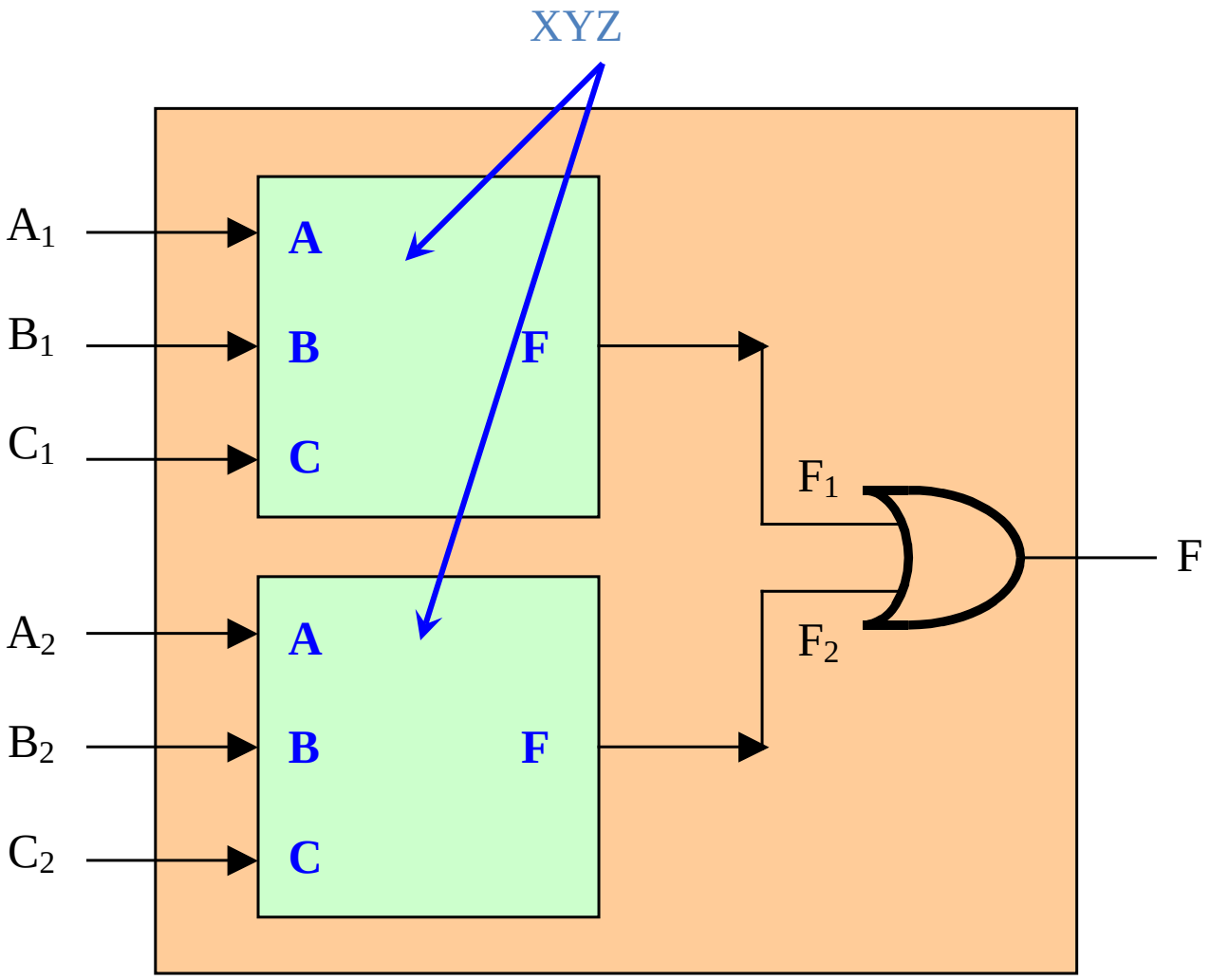
XYZ.VHD (All in One)

```
library ieee;
use ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.ALL;
--************************************
entity XYZ is
   port
   (
      A, B, C        :   in  std_logic;
      F              :   out std_logic
   );
end XYZ;
--************************************
architecture XYZ_arch1 of XYZ is
begin
   F <= (A and B) or (B and C) or (C and A);

end XYZ_arch1;
---------------------------------------
```

```
architecture XYZ_arch2 of XYZ is
begin
   F <= '1' when ((A = '1') and (B = '1')) else
        '1' when ((B = '1') and (C = '1')) else
        '1' when ((A = '1') and (C = '1')) else
        '0';

end XYZ_arch2;
---------------------------------------
architecture XYZ_arch3 of XYZ is
begin
   process (A, B, C)
   begin
      if (A = '1') and (B = '1') then
         F <= '1';
      elsif (B = '1') and (C = '1') then
         F <= '1';
      elsif (A = '1') and (C = '1') then
         F <= '1';
      else
         F <= '0';
      end if;
   end process;
end XYZ_arch3;
---------------------------------------
```

```
architecture XYZ_arch4 of XYZ
is
   signal F1, F2, F3: std_logic;
begin
   U1: and2
      port map(A, C, F1);
   U2: and2
      port map(A, B, F2);
   U3: and2
      port map(B, C, F3);
   F <= F1 or F2 or F3;
end XYZ_arch4;
---------------------------------------
```

Expanded Clock Diagram:



MY_PACKAGE.VHD

```
library ieee;
use ieee.std_logic_1164.all;
------------------------------
package my_package is
    component XYZ is
    port
    (
        A, B, C        :   in  std_logic;
        F              :   out std_logic
    );
    end component;
end my_package;
```

XXYYZZ.VHD

```
library ieee;
use ieee.std_logic_1164.all;
library work;
use work.my_package.all;
--------------------------------
entity XXYYZZ is
   port
   (
      A1, A2, B1, B2, C1, C2  :   in  std_logic;
      F                       :   out std_logic
   );
end XXYYZZ;
--------------------------------
architecture XXYYZZ_arch of XXYYZZ is
   signal F1, F2 : std_logic;
begin
   G1: XYZ
   port map(A1, B1, C1, F1);
   G2: XYZ
   port map(A2, B2, C2, F2);

   F <= F1 or F2;

end XXYYZZ_arch;
```