Digital Logic for Medicine and Biology research:

A hardware implementation of the Smith-Waterman algorithm for DNA comparison

Author: Javier Resano, Associate Professor,
Universidad de Zaragoza, Spain

Index

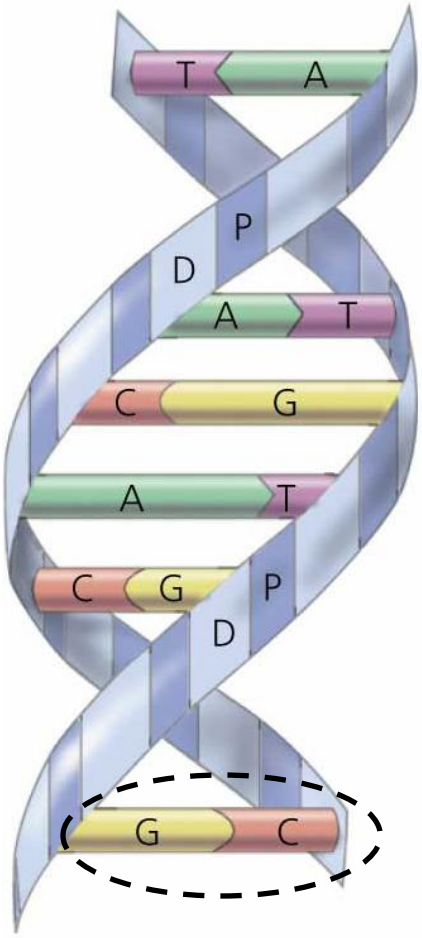
- Introduction to bioinformatics
- Options to speed up DNA comparisons
- FPGAs
- The Smith-Waterman algorithm
- Implementation details
- Overview of the project

Bioinformatics: studying the DNA

- Currently it is a major research field that directly influences Biology and Medicine
- Definition: *"The mathematical, statistical and computing methods that aim to solve biological problems using DNA and amino acid sequences and related information"*
- Why is DNA so important?

DNA stores the genetic information

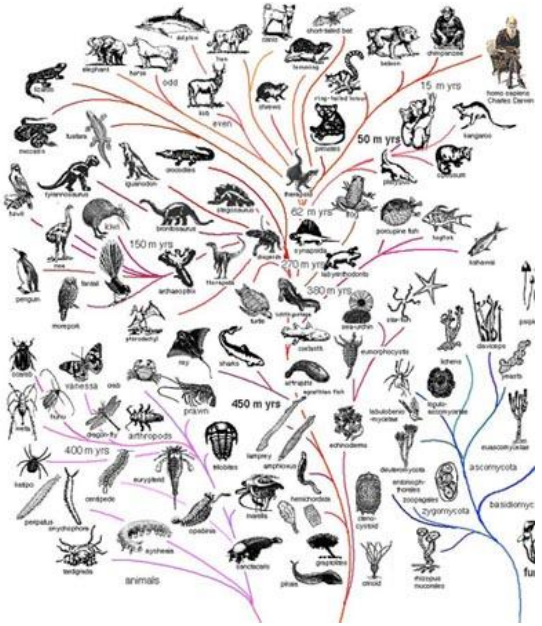
- DNA consists of two long polymers of simple units called nucleotides.
- Each nucleotide contains a nucleobase.
- The sequence of these bases is the key to store the genetic information.
- The genetic information is coded using only four different nucleobases:
  - adenine (A)
  - cytosine (C)
  - guanine (G)
  - thymine (T).



Pair of nucleotides with bases G and C

DNA comparisons are very useful for Medicine and Biology

- The DNA sequences of hundreds of organisms have been decoded and stored in databases
- In Biology this information is used to:
  - analyze the evolution of the species
  - study the biodiversity of a given ecosystem

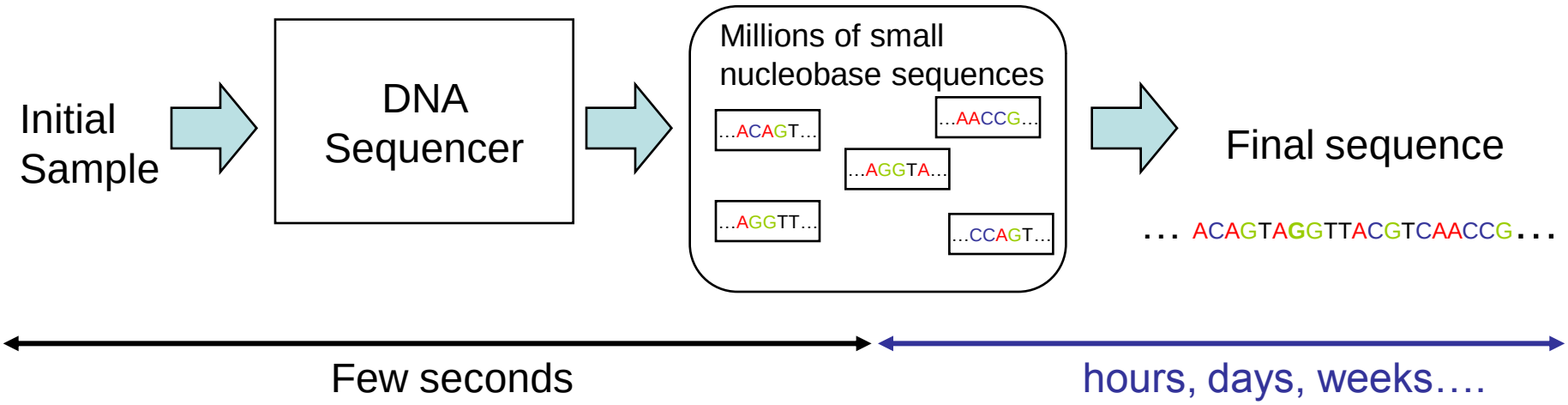


DNA comparisons are very useful for Medicine and Biology

- In Medicine, DNA analysis is used for many different purposes:
  - to find correlations between a given disease and DNA information
  - to analyze the cancer mutations
  - to study the evolution of viruses

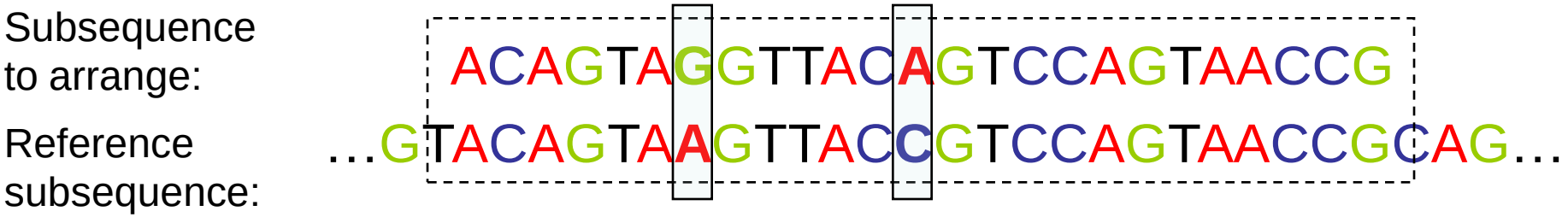
Obtaining the DNA sequence of a given organism

- The DNA sequence of a human being is composed of six billion nucleobases (three billions base pairs).
- The current most advanced technology can process a sample and obtain this information in just a few seconds.
- The output is not the DNA sequence needed for Medicine or Biology research, but millions of small sequences of 50-100 nucleobases



The intermediate sequences must be sorted and arranged

- This process is carried out using a reference DNA sequence.
- We can use this reference sequence because the difference in the DNA sequence of two individuals from the same specie are very small.
- If we find a subsequence almost identical to the one that we are trying to arrange (with only one or two different bases) we can assume that we have found the right location.
- To sort all the sequences we need to carry out billions of comparisons!
- And the output must report the degree of similarity between the two sequences. Normally this is called ***mutation-distance***.



Traditional solution: use supercomputers

- Supercomputer can speedup the process carrying out hundreds of comparisons in parallel.
- But they are **very expensive**: not all the researchers can access this useful infrastructure.
- Moreover, supercomputers are frequently shared among different research groups, and their use may become a bottleneck.

Mare Nostrum Supercomputer (www.bsc.es) used for the **human genome** research project but also for:

- protein research
- astrophysical simulations
- weather forecasting
- geophysical modelling
- …

We need more affordable options

- **The Human Genome Project (HGP)** was an international scientific research project with a primary goal to determine the sequence of chemical base pairs which make up human DNA.
- The Human Genome Project was completed in 2003 at an overall cost of **$3 billion** (US).
- It is clear that we need to find more affordable options.

• The **Archon X Prize for Genomics**, aims to dramatically reduce the time and cost of dealing with DNA sequences.

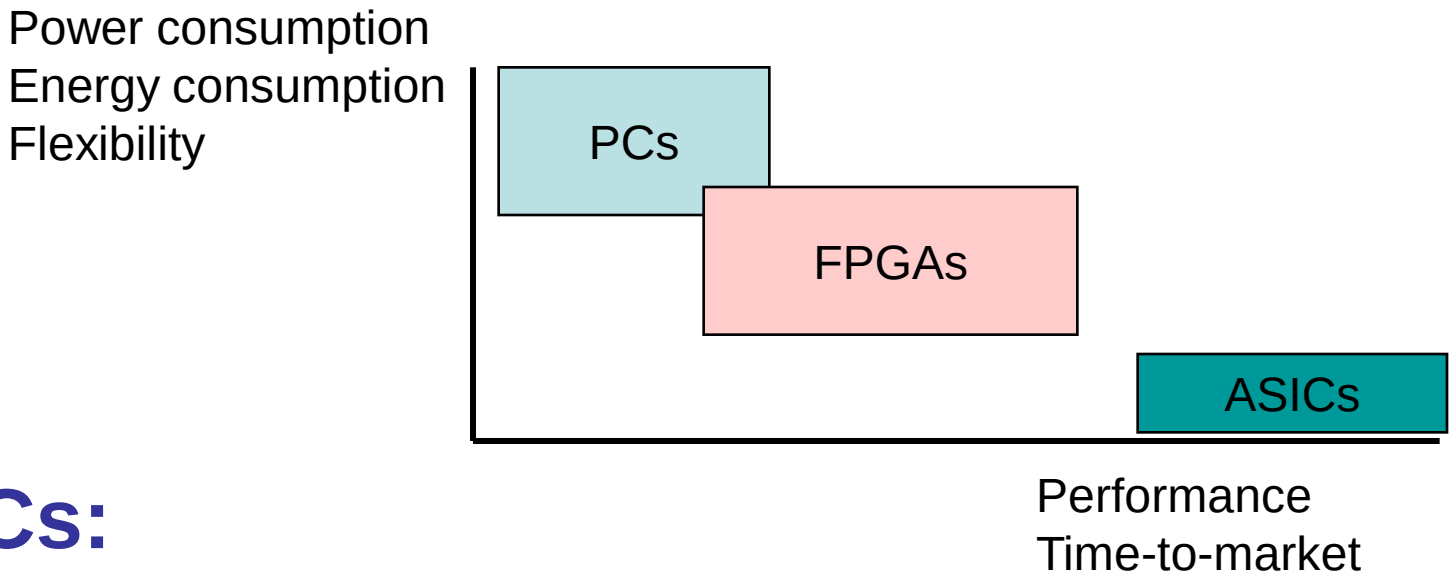
• The purpose is to enter in a new era of **personalized medicine.**

• The first Team that can sequence **100 human genomes** within **10 days** or less, with a platform that cost **less than $10,000 (US) per genome** will obtain a **$10 million prize** (US).

We need other platforms that provide the needed performance at a lower cost

- Option 1: Personal computers
- Option 2: Application Specific Integrated Circuits (ASICs).
- Option 3: Field-Programmable Gate Arrays (FPGAs)

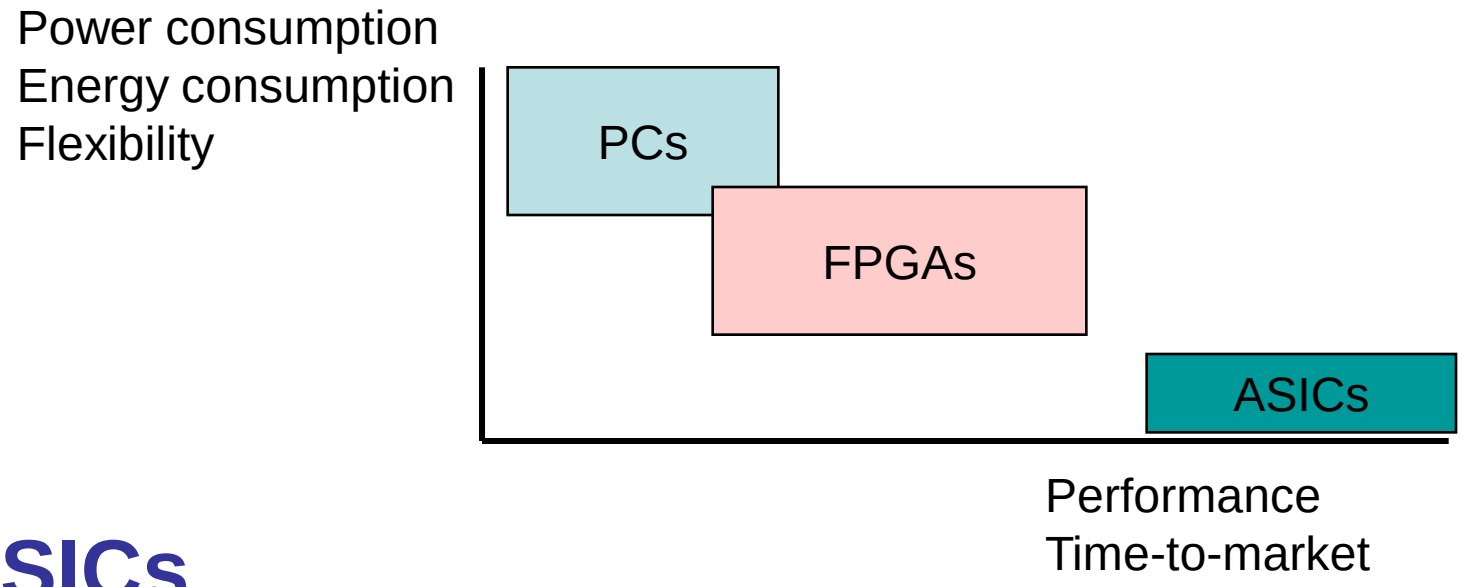
PCs, FPGAs and ASICs: Trade-offs



PCs:

- Fixed Hardware …
- …but can be used for any problem (great flexibility)
- Important overheads since the hardware has not been optimized for each given problem (reduced performance and important energy/power overheads compare to specific solutions)
- But lots of helpful tools are available (fastest Time-to-market)

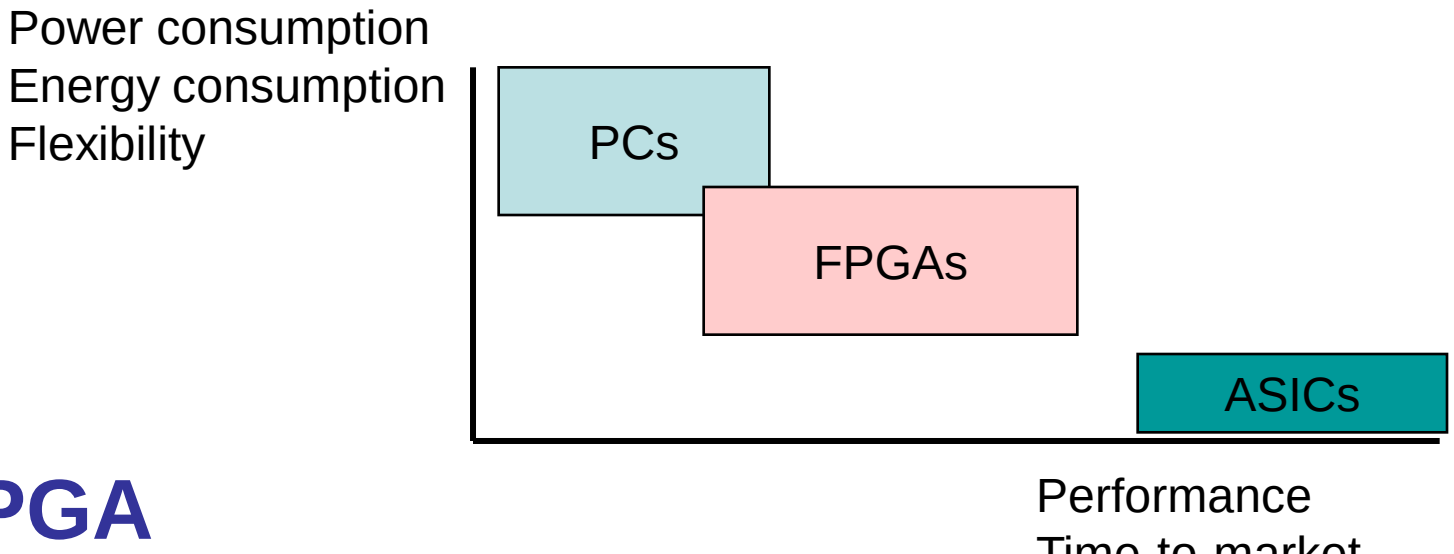
PCs, FPGAs and ASICs: Trade-offs



ASICs

- Custom Hardware Design that can be optimized for a given problem (great performance)
- No overheads due to unnecessary resources (low energy/power consumption)
- Extremely complex design-flow (it takes a lot of time to design an ASIC)
- Very expensive production process
- It can only be used for its specific purpose (no flexibility)

PCs, FPGAs and ASICs: Trade-offs



FPGA

- Intermediate solution
- Custom hardware design that is optimized for a given problem (good performance and energy/power efficiency)
- But based on fixed basic cells and existing routing resources (less efficient than ASICs for all metrics)
- Easy to test (better time to market than ASICs)
- Affordable price.

Good performance for an affordable price

FPGAs are programmable hardware devices

- FPGAs are arrays of simple programmable hardware components called "logic blocks”.
- FPGAs also include interconnection resources that support almost all possible connections among these blocks.
- Complex hardware designs can be obtained using simple “logic blocks” and connecting them.
- Current FPGAs also include additional resources such as simple processors, memory resources and multipliers.

FPGA structure

Array of logic blocks

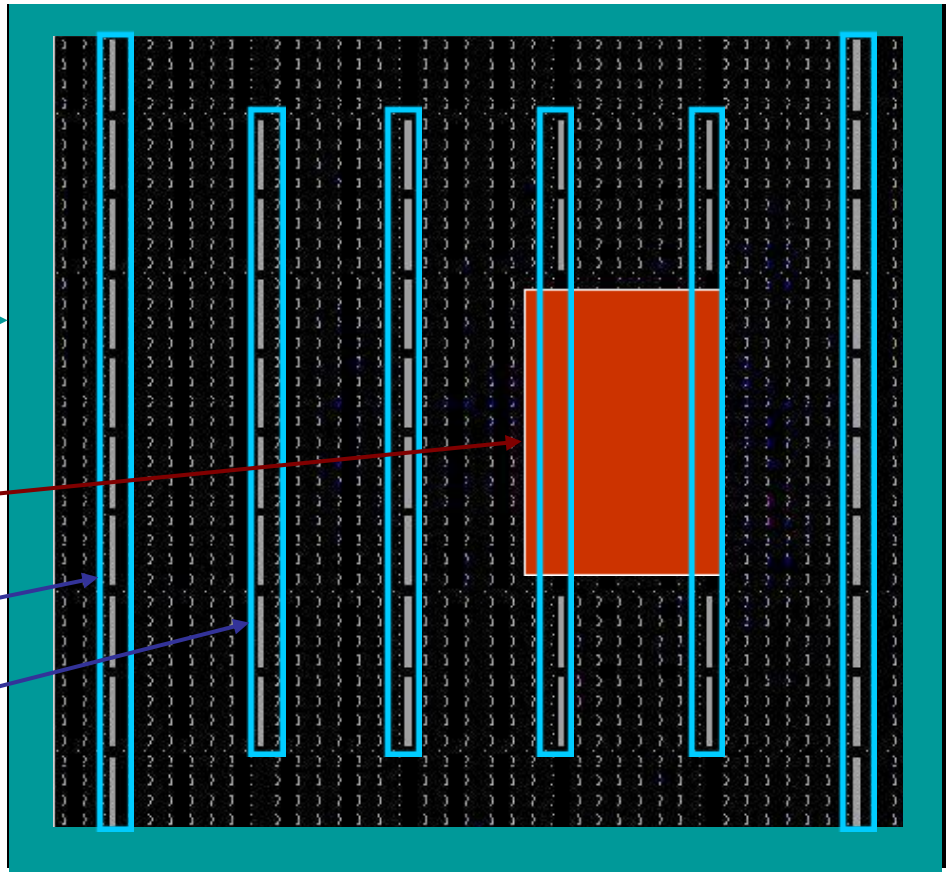
Interconnection resources

I/O

One or several processors
(available only in some FPGA families)

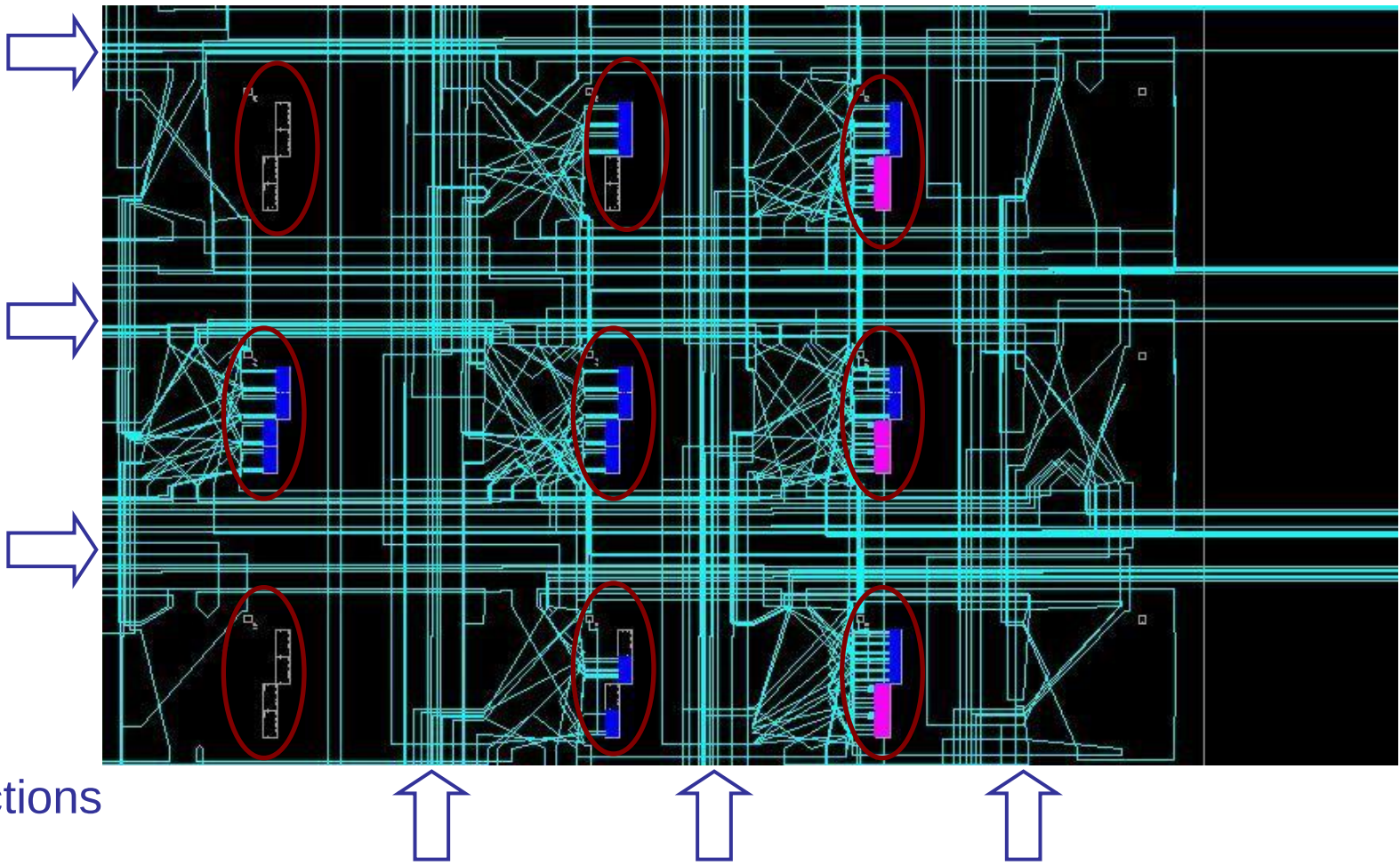
RAM memory

Multipliers

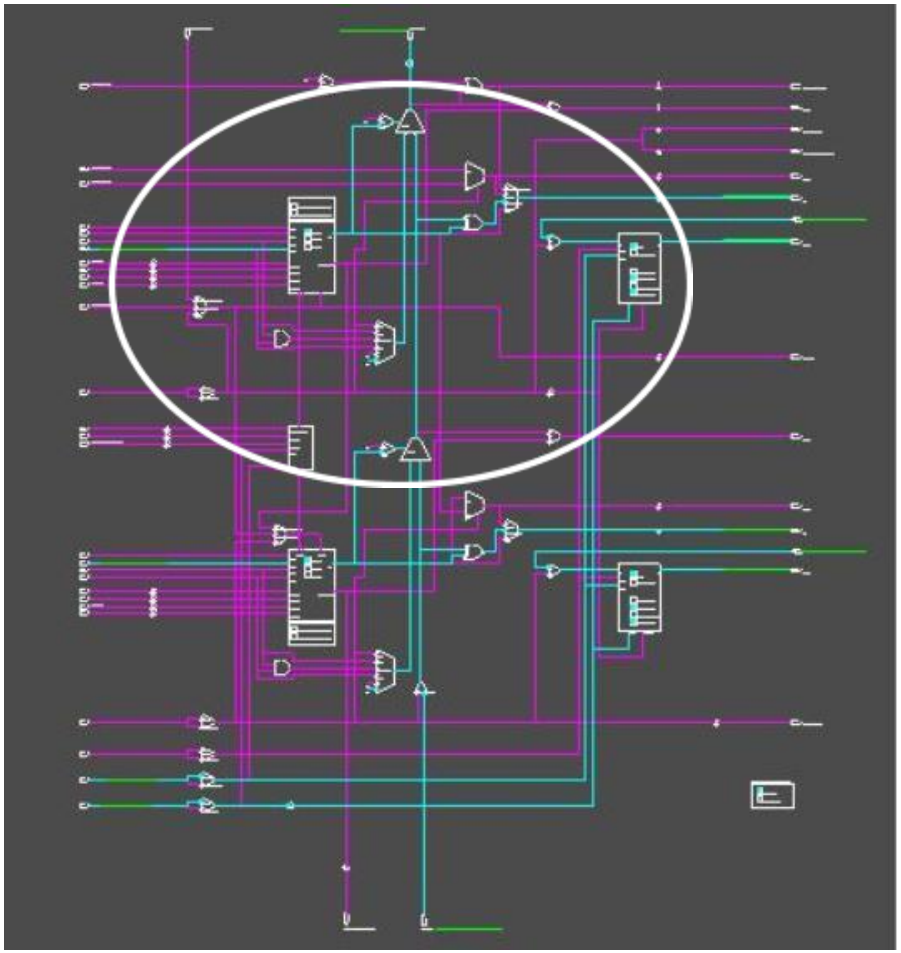


Configuration memory: distributed all over the FPGA, it is used to configure the FPGA to implement a given circuit

The main drawback of FPGAs is the excessive area overhead due to interconnections



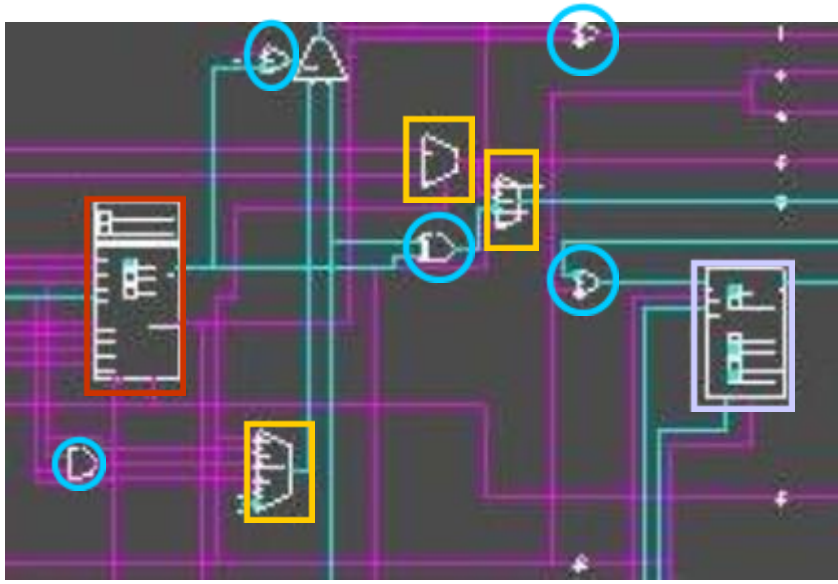
Structure of the logic blocks



Example of a logic block

LUTs

Logic gates

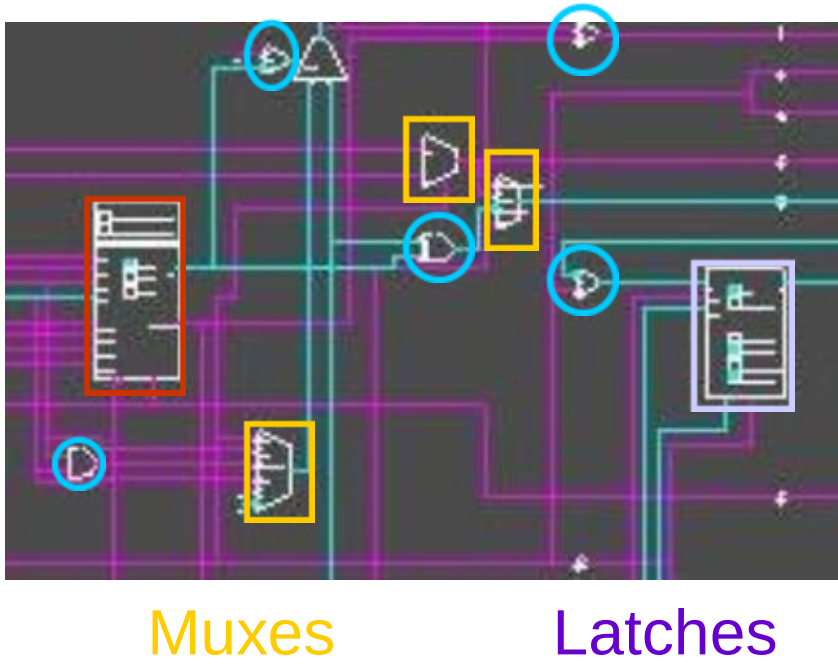


Muxes

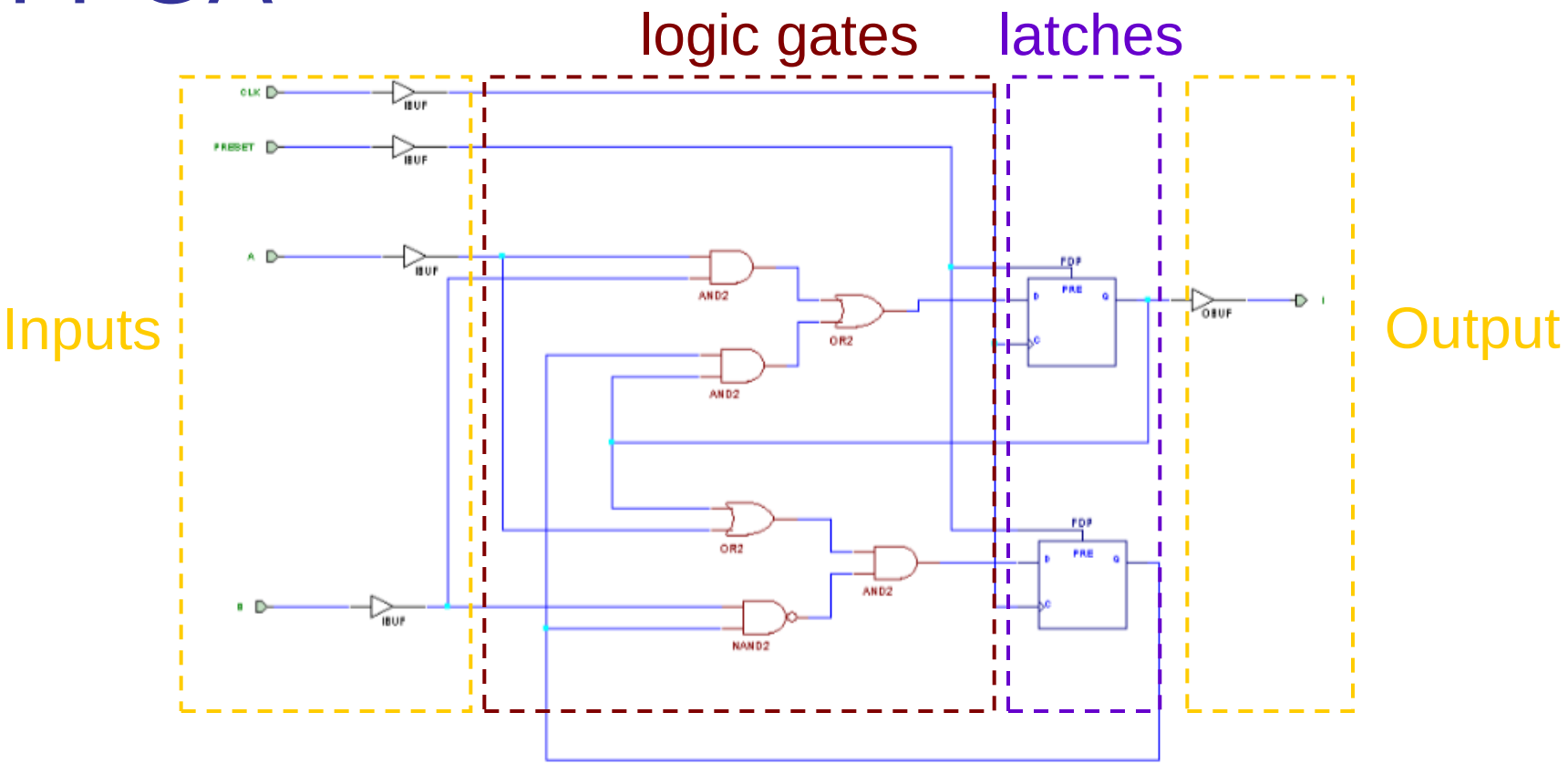
Latches

Structure of the logic blocks

- Muxes are used to select among different inputs
- Latches are used to store information
- Look up tables (LUTs) can be used both to store information or to implement basic logic functions.
- Complex logic functions are built using several LUTS



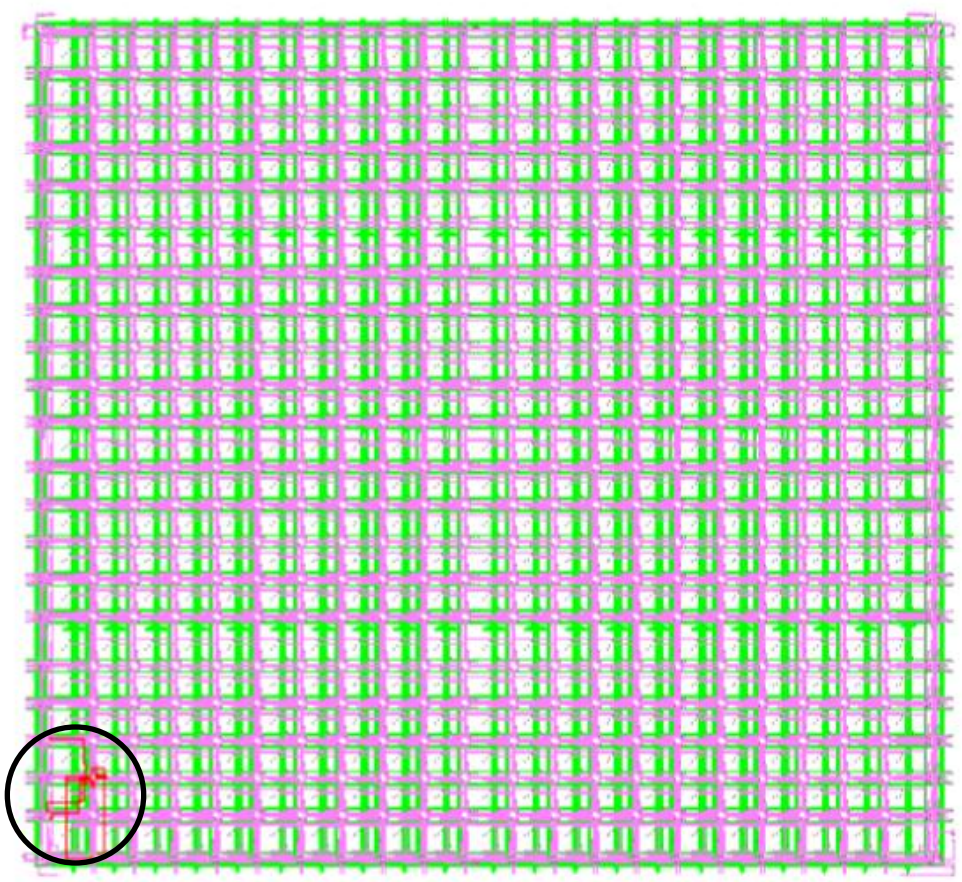
Implementing a simple circuit on an FPGA



First step: design the circuit

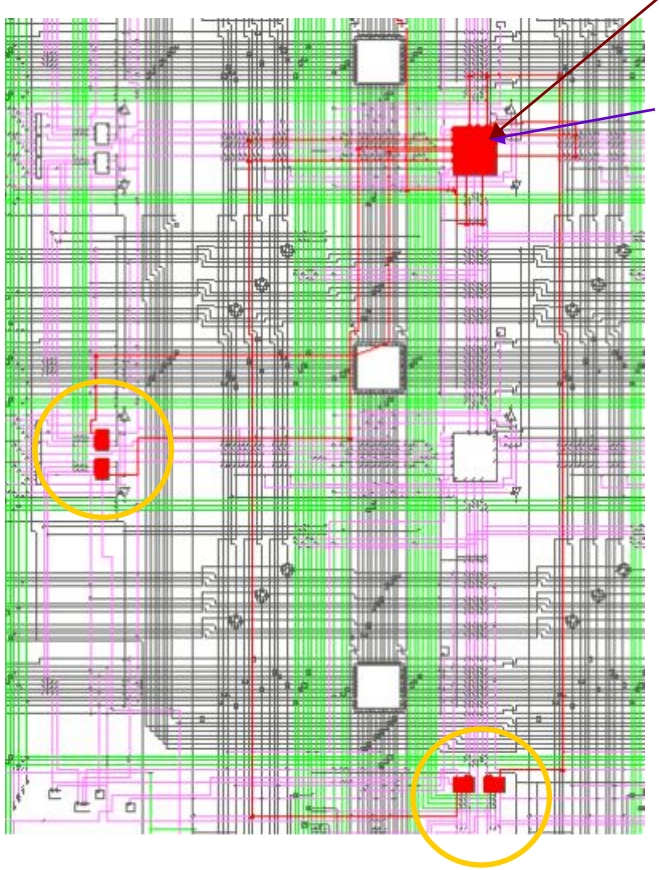
Second step: compile it for the FPGA

Implementing a simple circuit on an FPGA



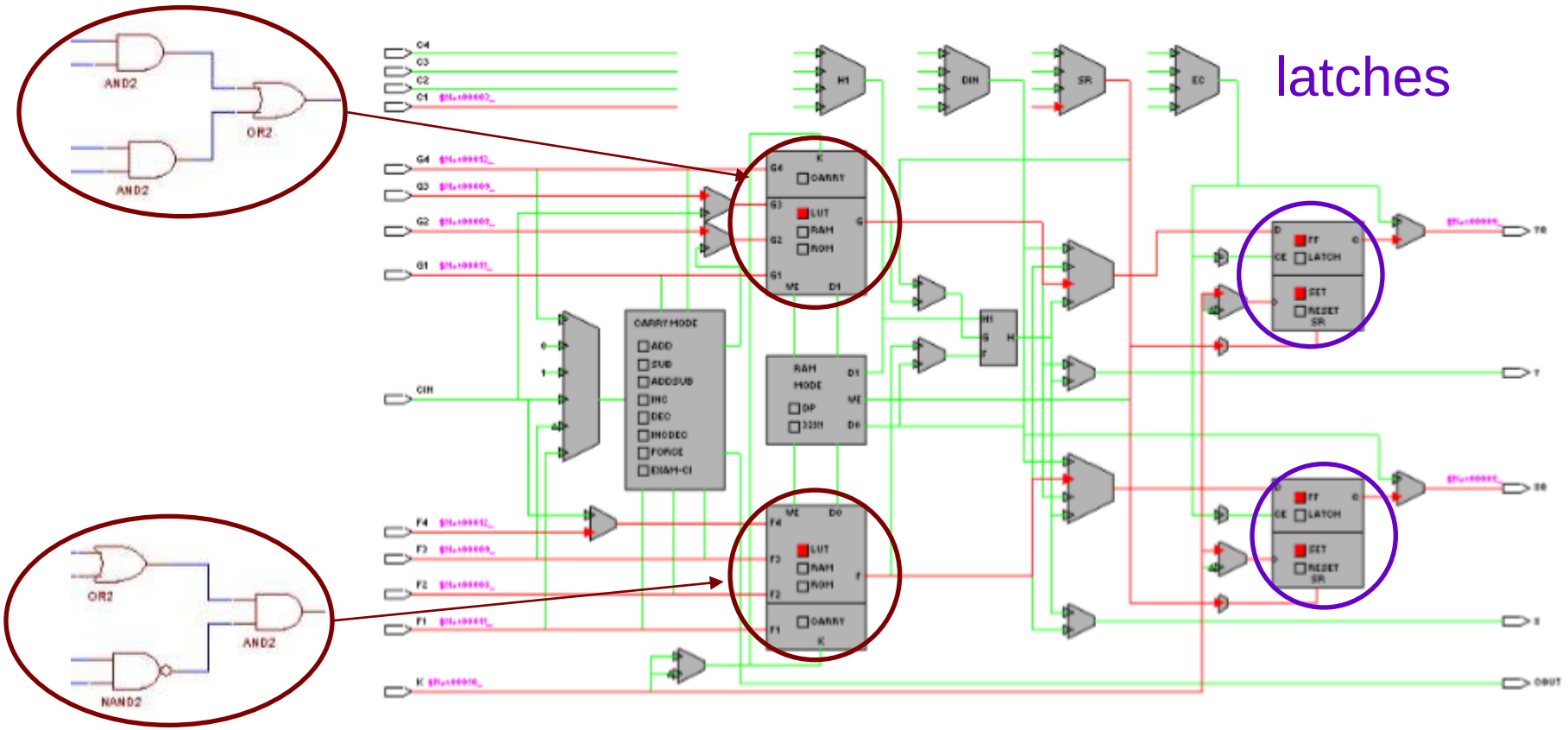
Region selected by the compiler to implement the circuit

Implementing a simple circuit on an FPGA



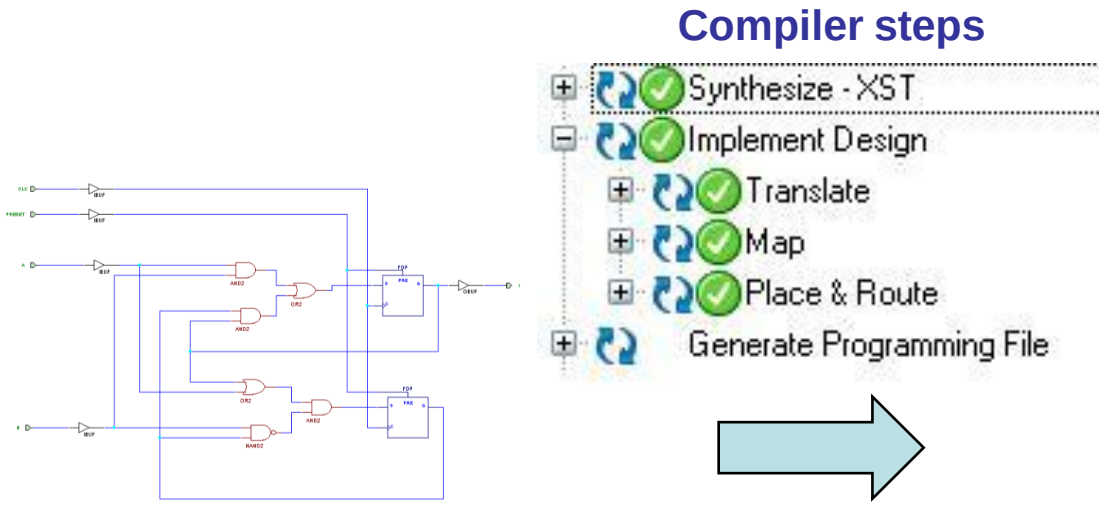
Implementing a simple circuit on an FPGA

Logic gates implemented using two look up tables



Steps to go from the original design to the FPGA implementation

1. Synthesize: checks the syntax of the input and assigns a hardware module to all their elements.
2. Translate: converts the initial input to an internal representation.
3. Map: fits the design into the available resources on the target FPGA (latches, LUTS, gates, and muxes).
4. Place & Route: assigns each component in the input to a physical component in the FPGA and generates the needed interconnection.
5. Generate Programming file: generates the configuration file for the FPGA.
6. Configure the FPGA: loads the configuration file on to the FPGA configuration memory.

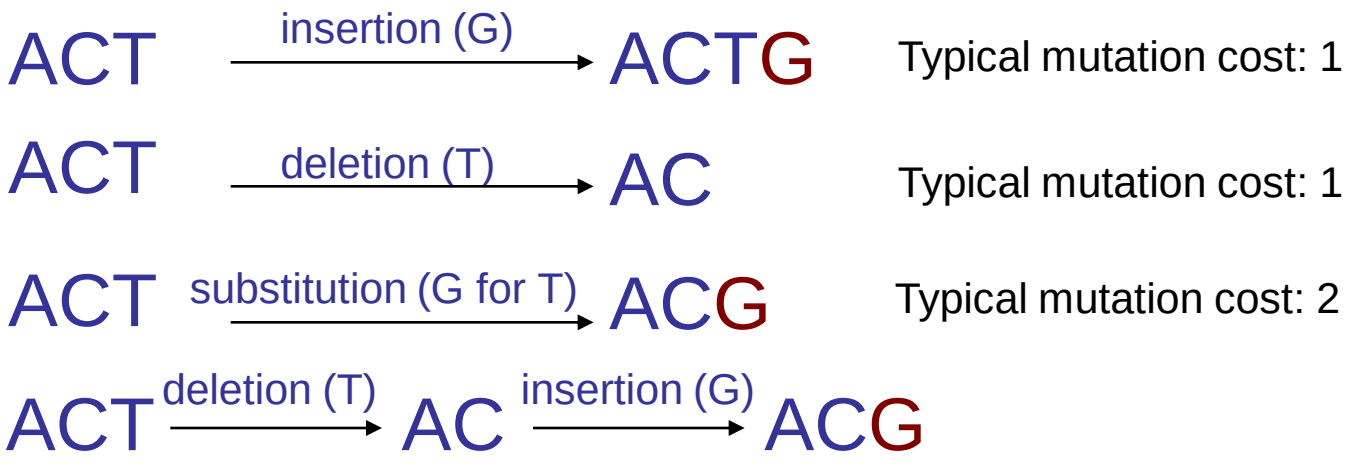


FPGAs for DNA comparison

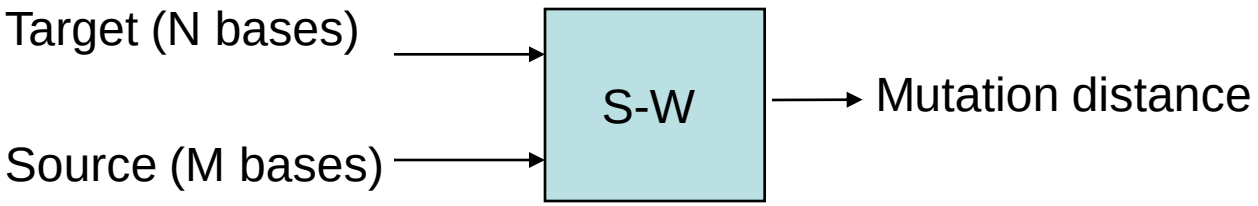
- Recently several research groups have proposed to use FPGAs to speed up DNA comparisons.
- And even some supercomputers have included FPGAs for this purpose:
  - The combination of the FPGA and the supercomputer is 26 times faster than an equivalent version executed only in the supercomputer (Accelerating the Smith-Waterman Algorithm on the Cray XD1, Margerm, Steve, and Maltby, Jim; Cray WP-0060406 2006).
- In this project we want to design a digital circuit that carries out DNA comparisons and to implement it on an FPGA
  - To simplify the assignment we will only implement the system for very small sequences.
  - But we will follow a modular approach that can be extended for sequences as long as needed.

The Smith-Waterman algorithm

- The Smith-Waterman algorithm is the most accurate method to compare DNA sequences
- It compares segments of these two sequences measuring the distance between each pair of segments
- The results of the previous comparisons are used to evaluate the distance of the new ones
- The distance between two sequences represents the mutation cost needed to go from one sequence (Source, S) to the other (Target, T)
- Three different mutations are considered:
  - insertion
  - deletion
  - substitution



Implementation of the S-W algorithm

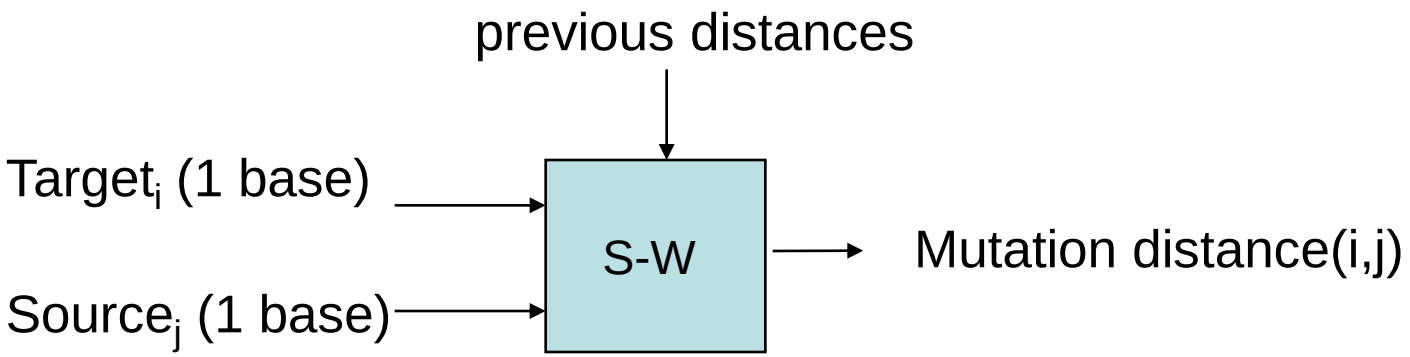


- In the inputs each base is codified with two bits (we need 2 bits per base because we have to codify 4 different bases)
- Assuming that the maximum size of the Target and the Source is 50 bases, we will need 200 bits for the inputs:
    - Do you know how can we start designing such a system?
    - how can we build a truth table of that size?

When a problem is too complex try applying a divide-and-conquer approach

1. Divide the complex problem into easier sub-problems
2. Solve those sub-problems
3. Compose the solution to the complex problem using the solutions of the sub-problems

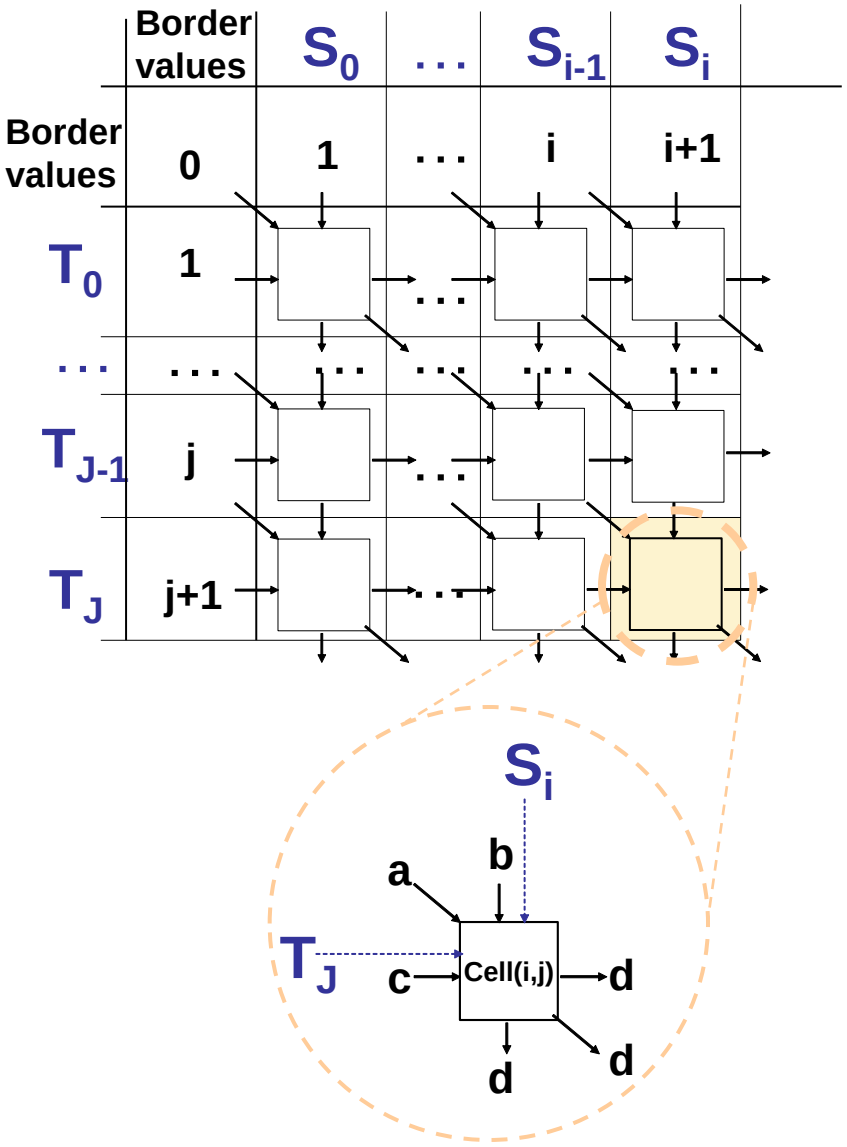
For the S-W algorithm we only need to solve one sub-problem, and then use it to build the complete solution



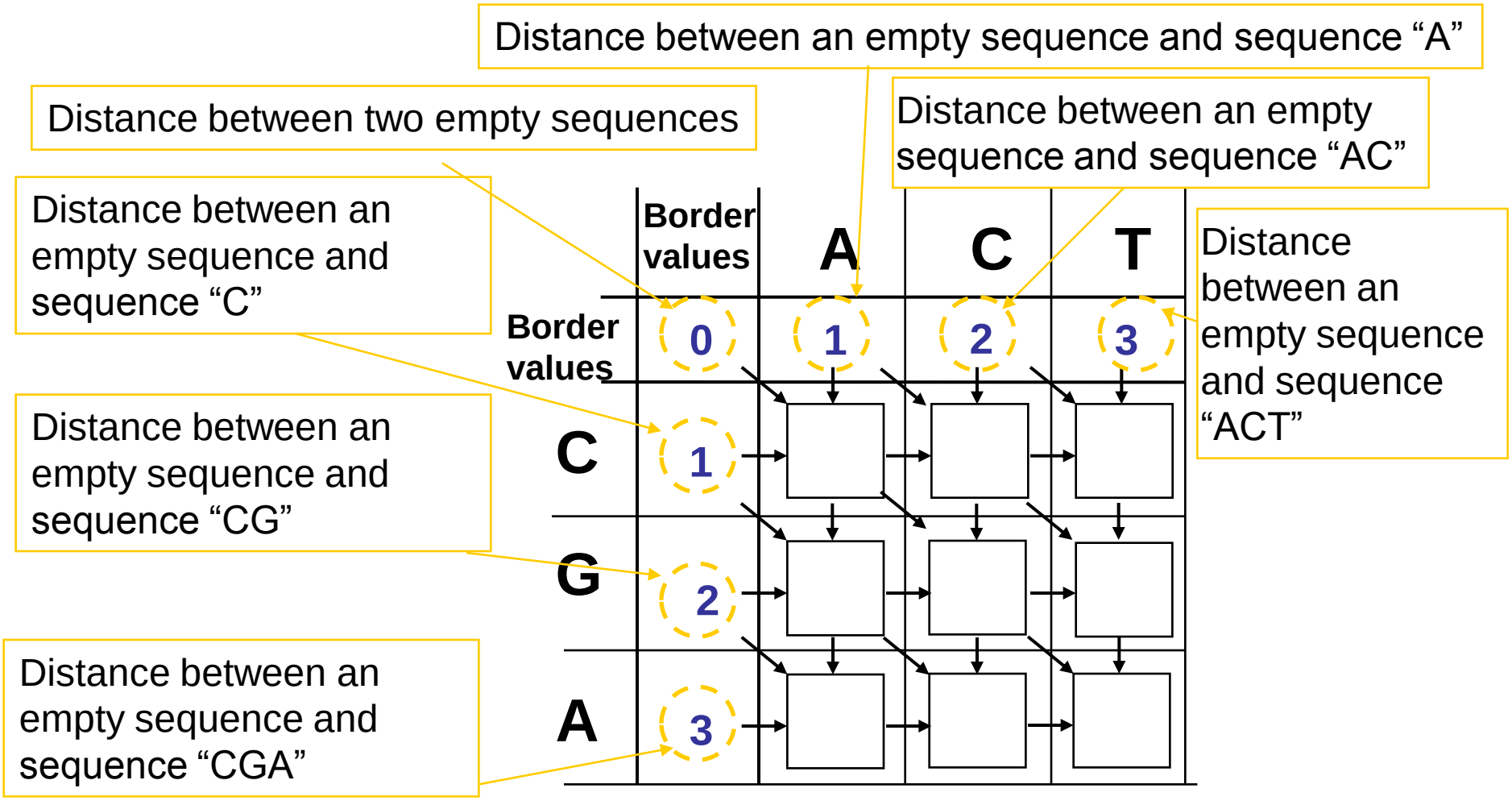
Implementation of the S-W algorithm

- 2D matrix of cells
- Each cell compares two sequences using:
  - the results of the previous cells or the border values (**a, b, c**)
  - two nucleobases, one from the Source (**Si**) and another from the Target (**Tj**)
- The output (**d**) is obtained using the following formulas:

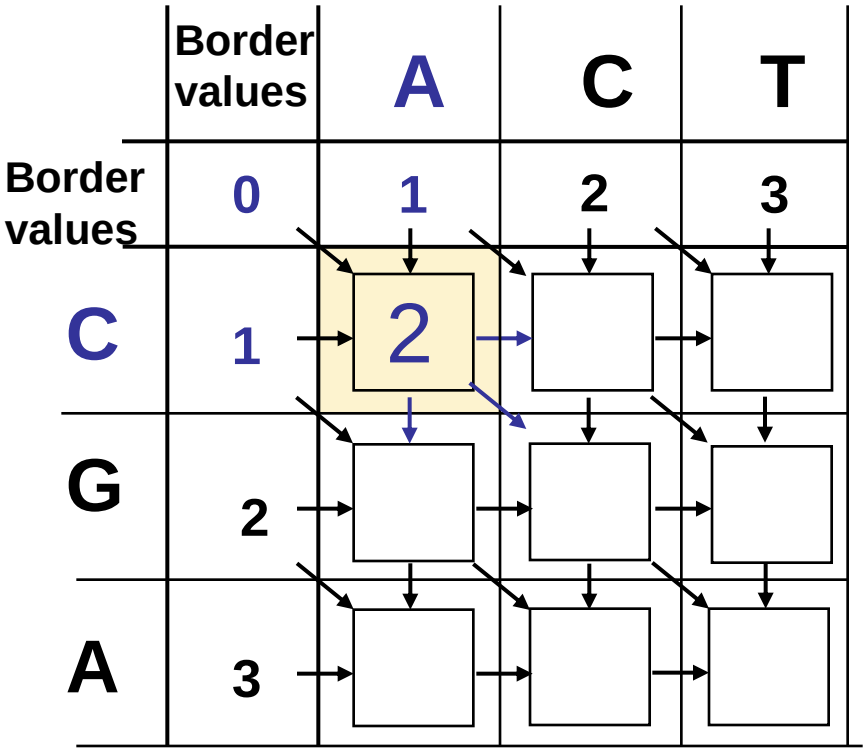
- IF ($S_i = T_j$) **a'**=a;
  ELSE **a'**=a+2;
- **d** = min (**a'**, **b** +1, **c** +1)



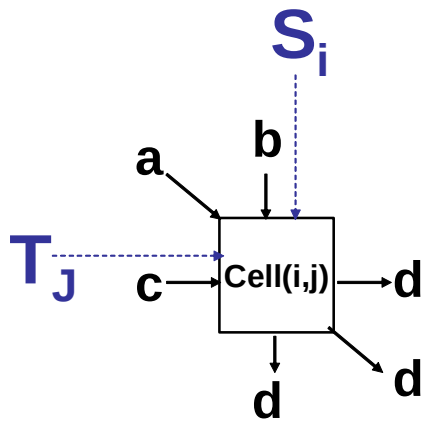
Meaning of the border values



Example: output of cell(0,0)



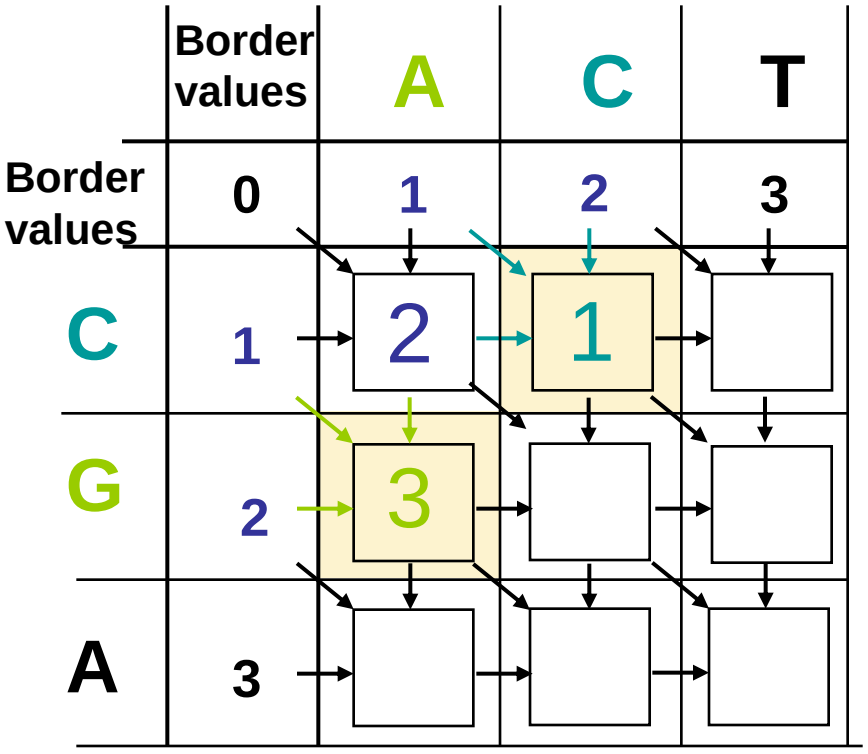
- Inputs:
  - a : 0
  - b : 1
  - c : 1
  - S_i : A
  - T_j : C

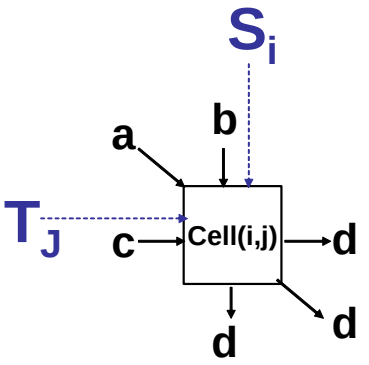


- **Output:**
  - A ≠ C → **a'** = a+2 = **2**;
  - **d** = min (2, 1+1, 1+1) = **2**

- IF (S_i = T_j) **a'**=a;
  ELSE **a'**=a+2;
- **d** = min (**a'**, **b** +1, **c** +1)

Example: outputs of cell(0,1) and cell(1,0)



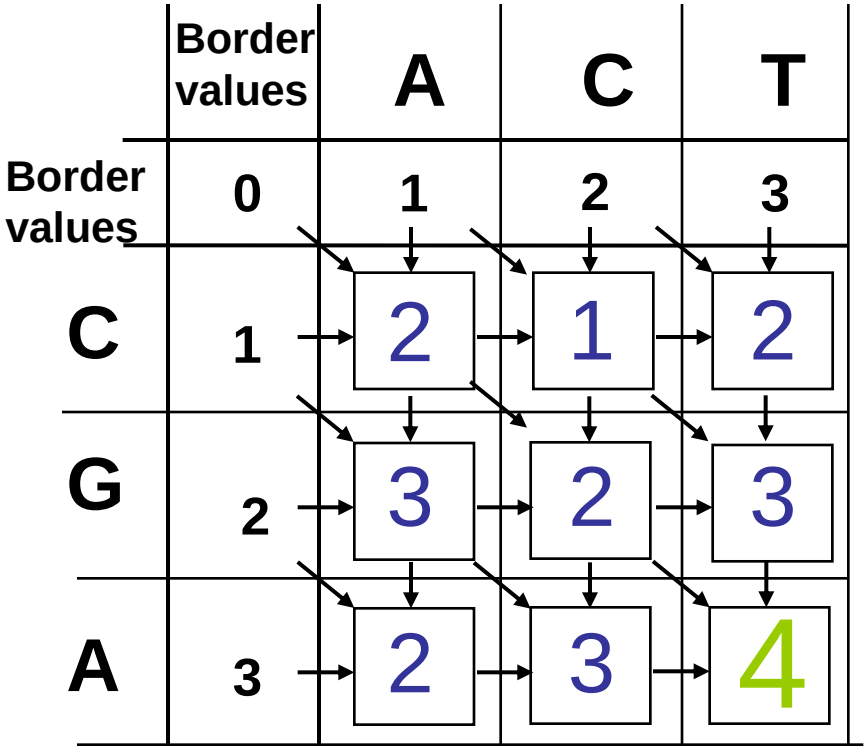


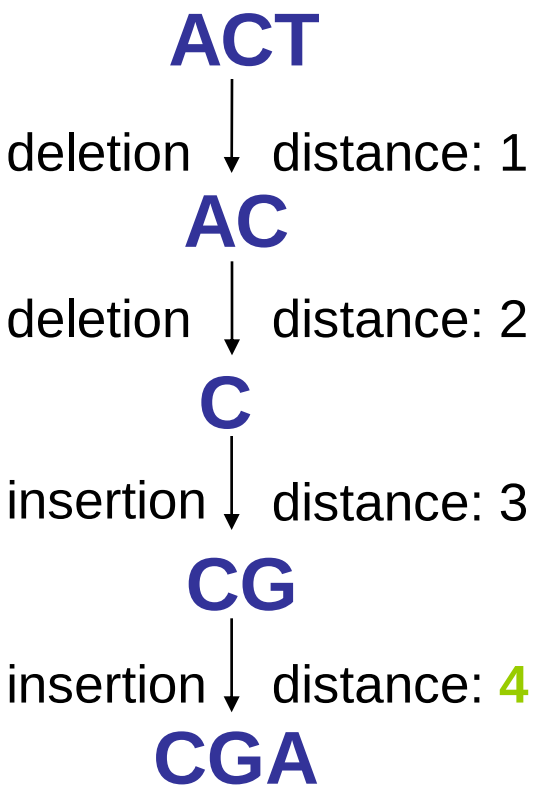
- **cell (0,1)**:
  - **Inputs**: a = 1; b = 2; c=2; T_0=C; S_1= C;
  - **Output:**
    - C = C → **a'** = a = **1**;
    - **d** = min (1, 3, 3) = **1**

- **cell (1,0):**
  - **Inputs**: a = 1; b = 2; c=2; T_0=G; S_1= A;
  - **Output:**
    - G ≠ A → **a'** = a+2 = **3**;
    - **d** = min (3, 3, 3) = **3**

- IF (S_i = T_j) **a'**=a;
  ELSE **a'**=a+2;
- **d** = min (**a'**, **b** +1, **c** +1)

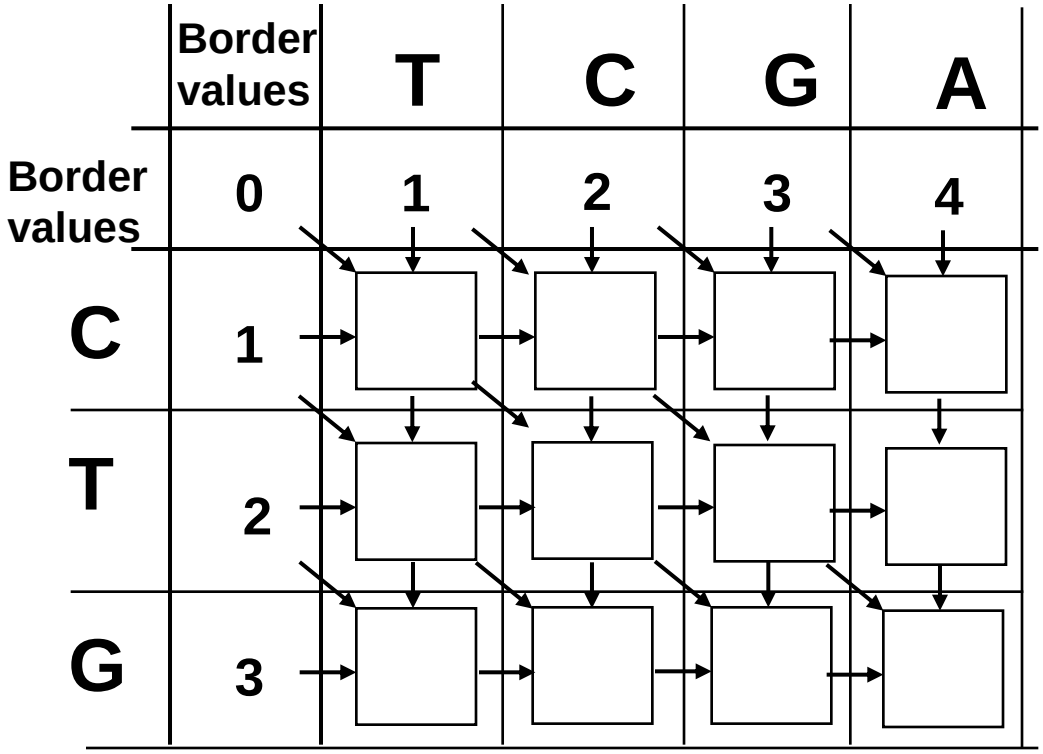
The final output represents the distance between the two sequences



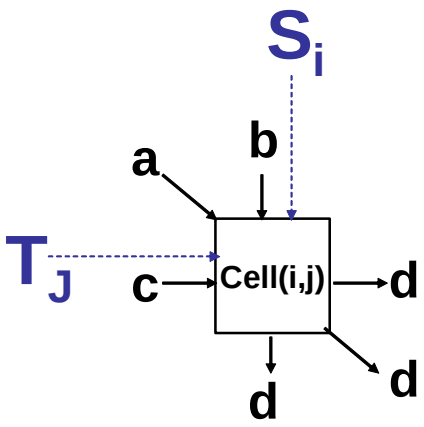


4 is the minimum distance to go from ACT to CGA

Do it yourself



TCGA



- IF ($S_i = T_j$) **a'**=a;
  ELSE **a'**=a+2;
- **d** = min (**a'**, **b** +1, **c** +1)

CTG

Designing the basic cell: still too complex

- Assuming that the distances can be encoded using three bits we will have 13 inputs (3 for each distance plus 2 for each base)

3-inputs truth table

a	b	c	z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

4-inputs truth table

a	b	c	d	z
0	0	0	0	0
0	0	0	0	1
0	0	0	1	1
0	0	0	1	0
0	0	1	0	0
0	0	1	0	1
0	0	1	1	1
0	0	1	1	0
0	1	0	0	1
0	1	0	0	0
0	1	0	1	0
0	1	0	1	1
0	1	1	0	1
0	1	1	0	0
0	1	1	1	0
0	1	1	1	1

5-inputs truth table

a	b	c	d	e	Z
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	1
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	1

a	b	c	d	e	Z
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	1

13-inputs truth table?

14 columns X
8192 rows

Processing that information is too complex

Designing the basic cell: still too complex

-Solutions:

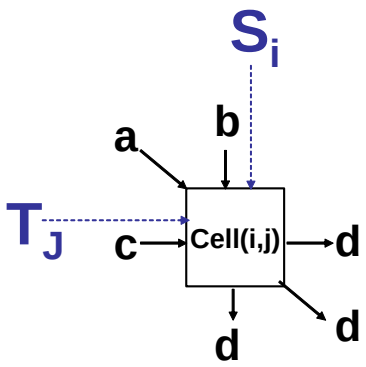
- Apply again a divide-and-conquer approach
- Simplify the design process using a modular approach
- If still some sub problems are too complex apply again a divide-and-conquer approach to them (this approach is called hierarchical design)

- These three ideas are the key of any real-world digital logic design

The modular approach: Reuse is the key for an efficient design-flow

- Some hardware modules can be found almost in every design
- It makes no sense to design them every time
- Reusing blocks from previous designs can drastically speed up the design process
- Design environments often include libraries of hardware components that can be directly included in your design
- Designers can include new components in the libraries

Designing the basic cell: identify the hardware components needed



- IF ($S_i = T_j$) **a'**=a;
  ELSE **a'**=a+2;
- **d** = min (**a'**, **b** +1, **c** +1)

(Si = Tj) → 2-bit comparator

a+2 → adder

a'= select between a and a+2 → We need three 2:1 muxes

b +1 → adder

c +1 → adder

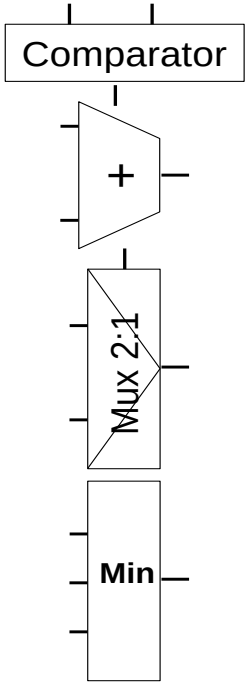
Available in all the schematics libraries

d = min (**a'**, **b** +1, **c** +1) → Not available in the library

Too complex: apply again a divide-and-conquer approach

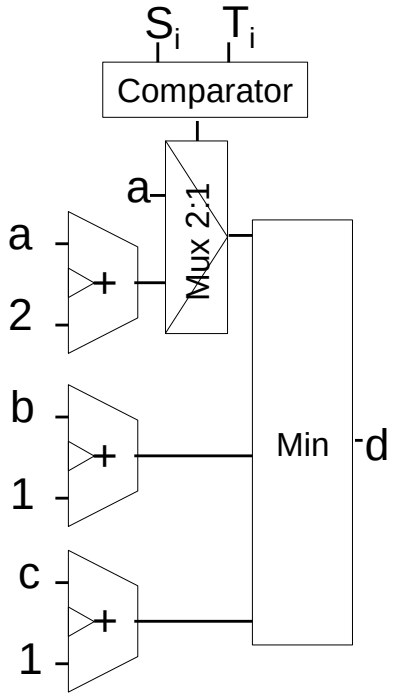
Evolution of the design

Step 1: Identifying the basic modules



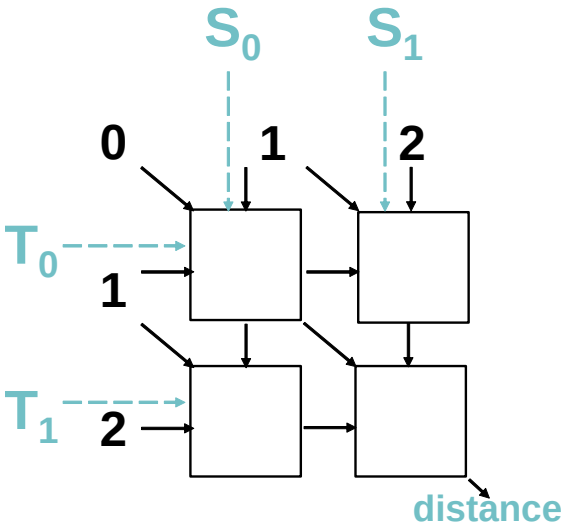
Include the Min module in the schematic library

Step 2: Designing the basic cell



Design the basic cell and include it in the sch. library

Step 3: Designing a 2x2 matrix



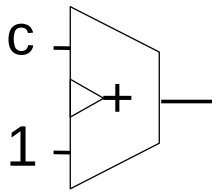
Final design

A three-level hierarchical design

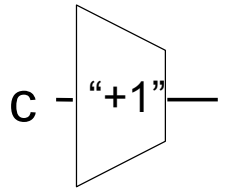
modular design vs. custom design: trade-offs

- In this project we will try to reuse as much as possible the logic modules available in the schematic library.
- Is this always the best approach?
  - Depend on your design constraints (i.e. area, performance constraints):
    - If you can meet your constraints with the existing modules this is definitely the best option.
    - Otherwise, you may consider spending sometime designing your own custom modules.
- Example: a module that adds one to a given number can be faster and smaller than a full adder.

Option 1: Use an existing adder
Pros: you do not need to design it
Cons: you are using more HW than needed



Option 2: Design a “+1” custom module
Pros: faster and smaller
Cons: you need to design it and test it



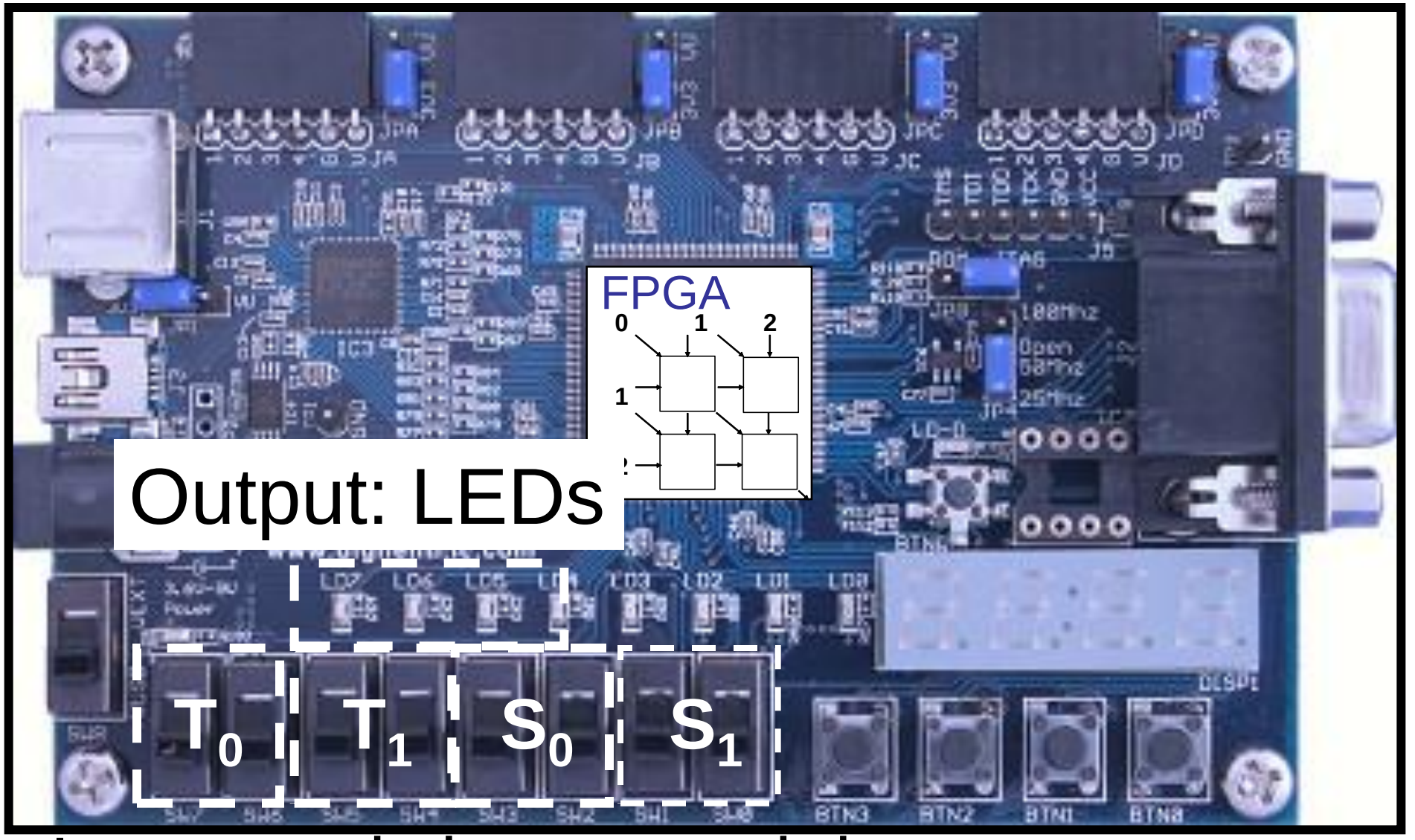
Testing is as important as designing

- In real-world designs more than 80% of the time is used for the simulation and test process

- We are responsible for our designs:
  - The design is not finished until we have exhaustively tested it

- We will test our design at two different levels:
  - Step 1: test the design using a software simulator.
  - Step 2: test the design in the FPGA

Implementation on an FPGA of the 2x2 matrix



Inputs: switches or push buttons

Conclusions

- Bioinformatics is one of the most active research fields and it influences both biology and Medicine
- DNA comparison is one of the basic operations of bioinformatics but it is very computational intensive
- FPGAs are an affordable solution to speed up these comparisons
- Using our knowledge in digital logic we will design a circuit that implements the S-W algorithm for DNA comparisons
- We will follow a divide-and-conquer approach, working at three different levels (hierarchical design) and reusing standard modules (modular design)
- We will implement and test a small system on an FPGA
- Finally we will evaluate our solution for larger systems using a theoretical model