Summary Lecture
Digital Logic for Medicine and Biology Research:

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

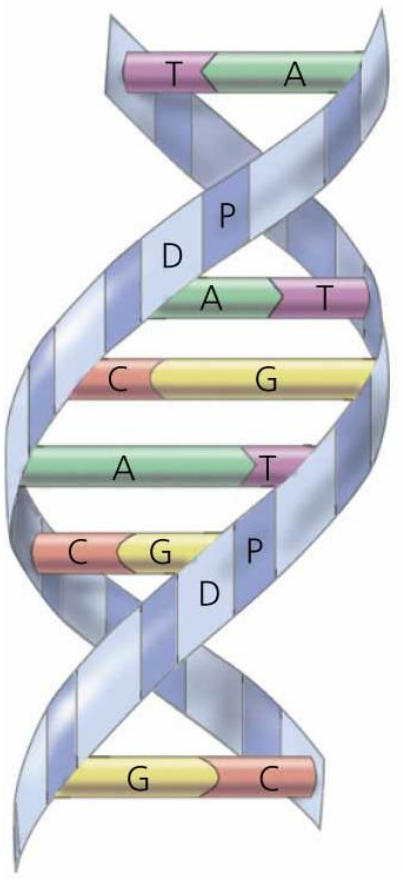
Author: Javier Resano, Associate Professor,
Universidad de Zaragoza, Spain

Index

- Digital logic Design for Bioinformatics
- Selecting the platform: why FPGAs?
- Outline of the design flow
- Evaluating our design
- Conclusions

Bioinformatics: studying the DNA

- Bioinformatics is a major research field that directly influences Biology and Medicine
- 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 and to study the biodiversity.
- 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



Bioinformatics demands High Performance Computing

- All these important research activities demand high-speed DNA sequence comparisons. For example:
  - comparing a given DNA sequence with the sequences stored in large scientific databases.
  - or trying to sort several millions of small sequences obtained in the laboratory by comparing them with a reference DNA sequence.
- Traditionally only supercomputers could carry out this comparisons (and they may need several weeks to do it)

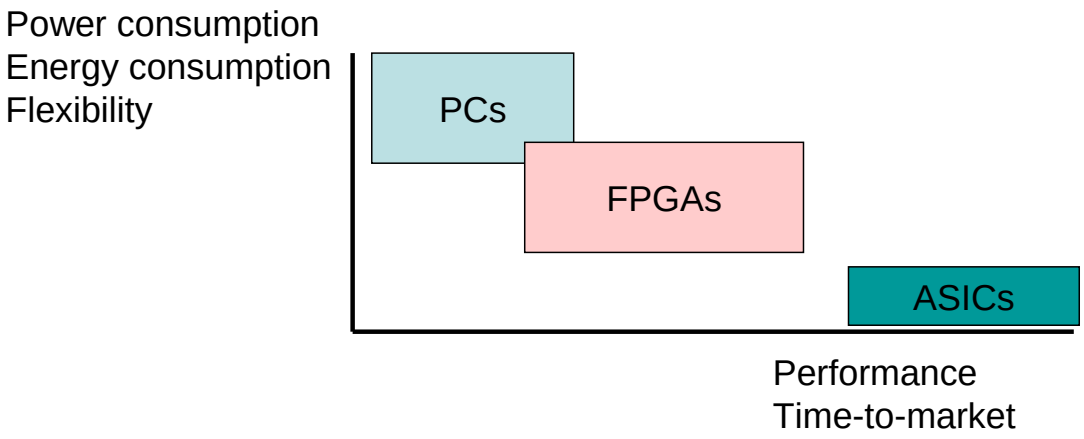
We need other platforms that provide the needed performance at a lower cost

- Option 1: Personal computers
- Option 2: Use digital logic design techniques to develop efficient hardware platforms for the Bioinformatics computations:
  - Application Specific Circuits (ASICs)
  - Field-Programmable Gate Array (FPGAs)
- These two options provide different trade-offs

Index

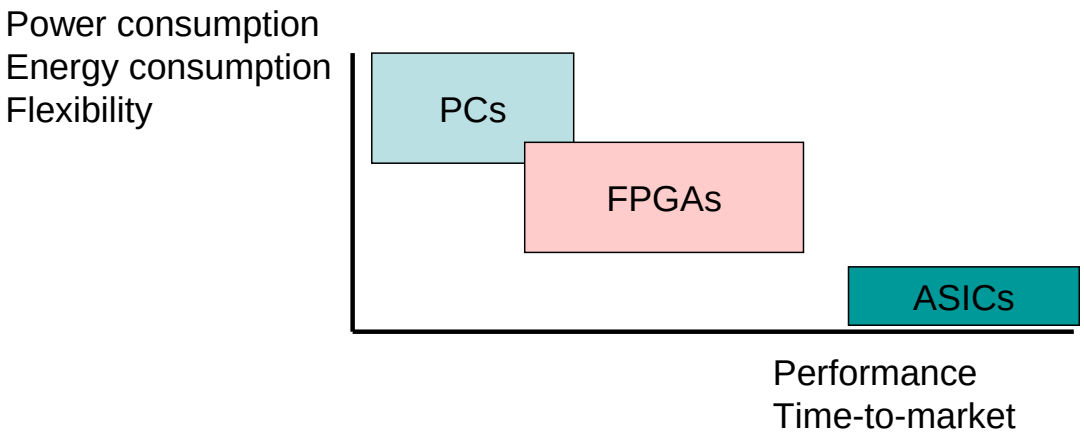
- Digital logic Design for Bioinformatics
- **Selecting the platform: why FPGAs?**
- Outline of the design flow
- Evaluating our design
- Conclusions

PCs, FPGAs and ASICs: Trade-offs



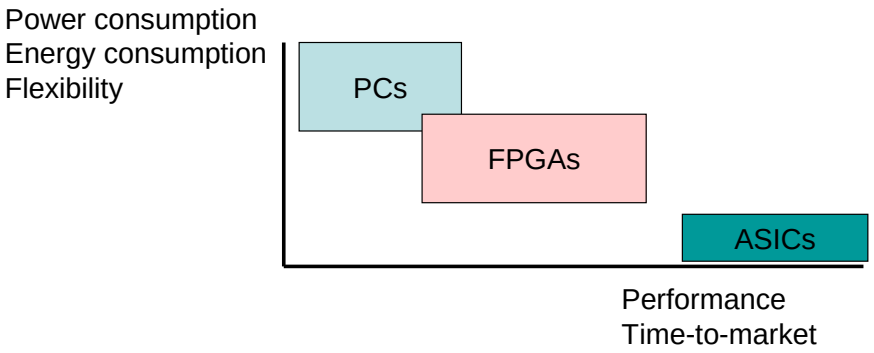
- Why FPGAs and ASICs provide better performance than PCs?
  - PCs have a fixed set of HW resources that can be used for any problem
  - When you design a digital logic circuit (for an FPGA or an ASIC) you can include customized hardware resources:
    - Our basic cells include four comparators and three adders.
    - In our 2x2 matrix we have 16 comparators and 12 adders working at the same time.

PCs, FPGAs and ASICs: Trade-offs



- Why FPGAs time-to-market is worst than PCs' time-to-market?
  - When developing a program for a PC the programmer does not need to know the hardware resources of the processor: the compiler and the operating system will do most of the hard work.
  - Compilers for HW platforms also do a lot of hard work, but still the designer has to work at a lower abstraction level:
    - In a PC you only have to use the hardware but in an FPGA you have to design it.

PCs, FPGAs and ASICs: Trade-offs



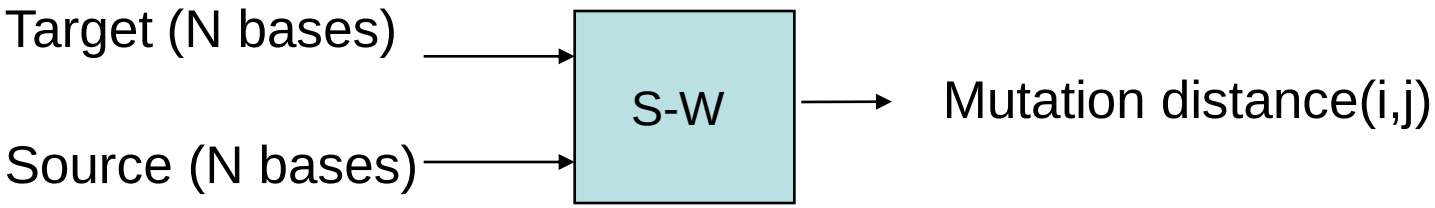
- Why we have selected an FPGA for this project if ASICs provide better performance?
  - ASICs can be completely customized for a given problem (great performance) but the design-flow is very complex.
  - FPGAs are an intermediate solution:
    - You can customize your hardware for a given problem (good performance) using the fixed FPGA resources.
    - Easy to test: you can configure the FPGA as soon as you have implemented your design.
    - Affordable price.
    - Good performance for an affordable price.

Index

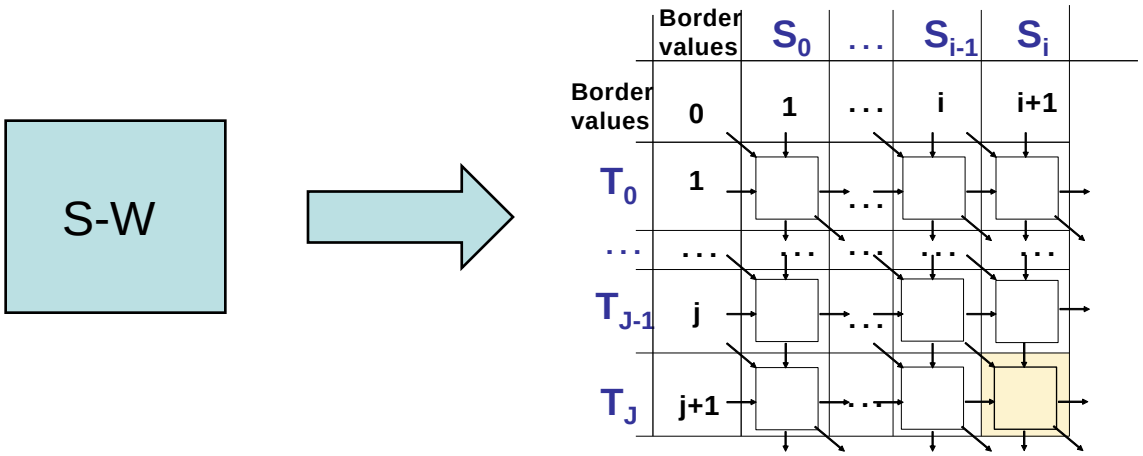
- Digital logic Design for Bioinformatics
- Selecting the platform: why FPGAs?
- **Outline of the design flow**
- Evaluating our design
- Conclusions

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

Initial problem:



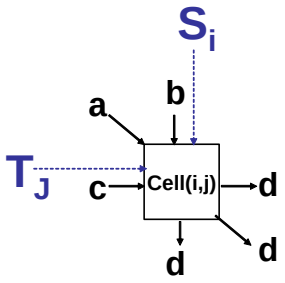
Too complex: Divide the problem into easier sub-problems



N*N identical small basic cells

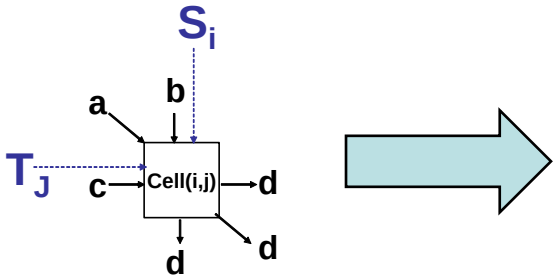
When a problem is too complex try applying a divide-and-conquer approach

Basic cell:



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

Still too complex: divide again the problem into easier sub-problems



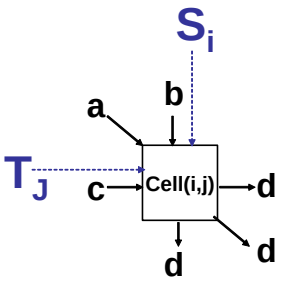
a+2 | **b** +1 | **c** +1 | (Si = Tj)

a'= select between a and a+2

d = min (**a'**, **b** +1, **c** +1)

Follow a modular approach: Reuse existing modules

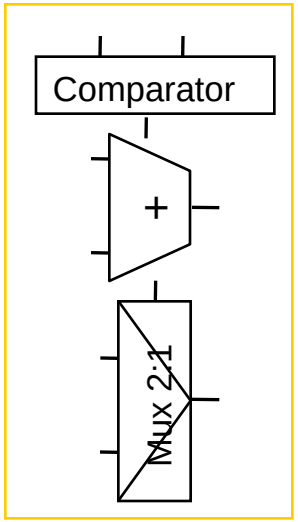
Basic cell:



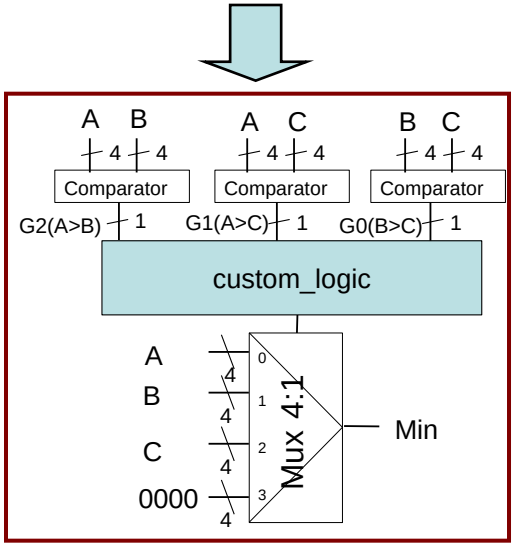
a+2 | **b** +1 | **c** +1 | (Si = Tj)

a'= select between a and a+2

Available in all the schematics libraries



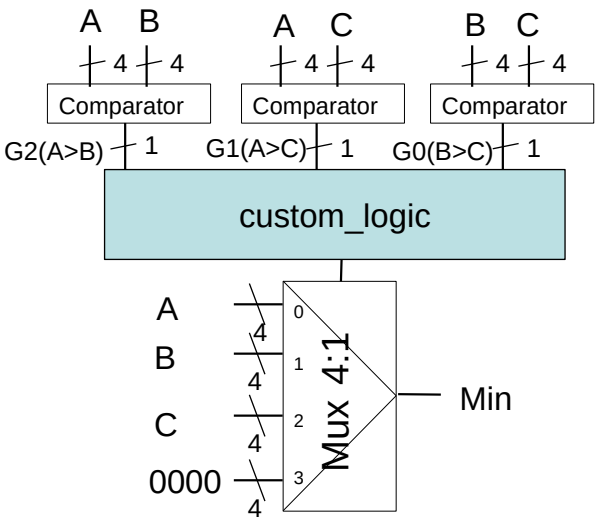
d = min (**a'**, **b** +1, **c** +1)



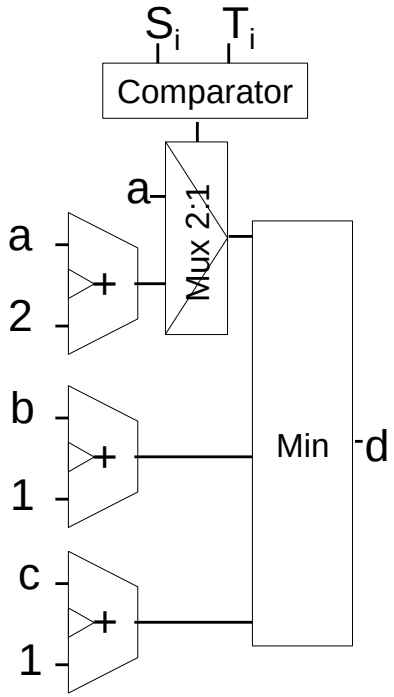
Too complex:

- Apply again a divide-and-conquer approach
- Identify the modules that can be reused from the schematic library
- Design the custom logic needed
- Assemble all the modules
- Generate a symbol and include it in the library

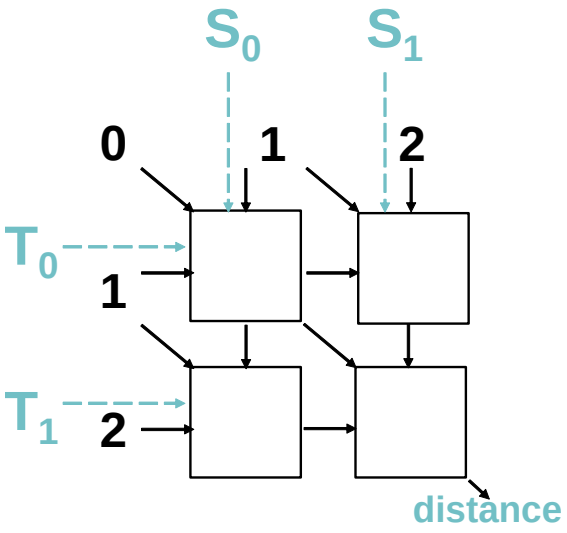
Hierarchical design: solve the sub-problems at the lower levels and use those solutions in the following level



Min module



Basic cell



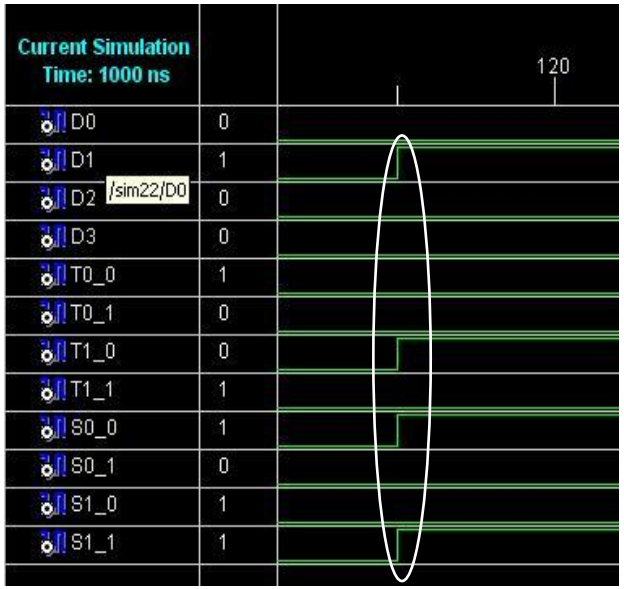
Final design

The design is not finished until it has been properly tested

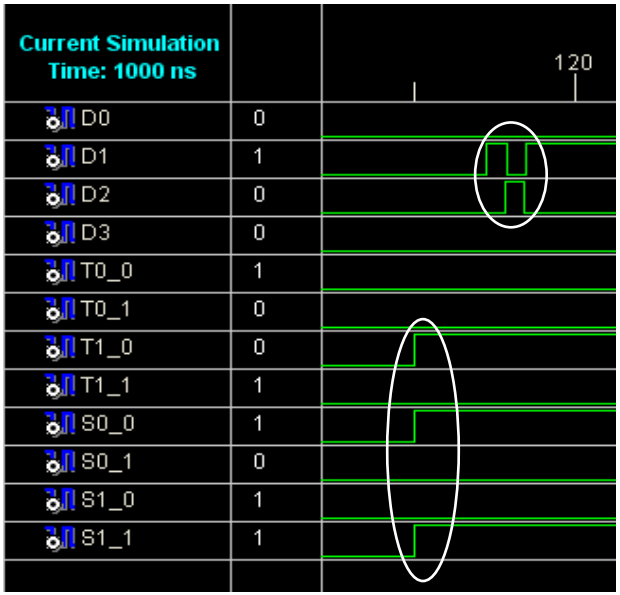
- In real-world designs more than 80% of the time is used for the simulation and test processes.
- We are responsible for our designs.
- We will test our design at three different levels:
  - Behavioral or Functional Simulation:
    - Simulates the design without taking into account the delays
    - It is not completely accurate but it is fast and platform independent
  - Post-Place & Route Simulation:
    - Includes a very accurate timing model generated during the implementation process
    - Slow and platform dependent
  - Finally: test the design in the FPGA

Functional Simulation vs. Post-Place & Route Simulation

Functional simulation



Post-Place & Route simulation



Functional Simulation: when the inputs change the outputs are immediately updated

Post-Place & Route Simulation:

- when the inputs change the output is updated after a delay
- each output has a different delay: this generates temporal wrong outputs

Index

- Digital logic Design for Bioinformatics
- Selecting the platform: why FPGAs?
- Outline of the design flow
- **Evaluating our design**
- Conclusions

Evaluating the design of a NxN matrix

- We have designed, implemented and tested a very small module for DNA comparisons
- But our modular design can be easily extended to compare larger sequences
    - Which modifications are needed in order to design a NxN matrix?
    - How efficient our design will be for larger sequences?

Modifications for a NxN matrix

1. Our design codifies the distances using 4 bits. But for larger matrices this may not be sufficient:
   - Hence we should use more bits not only to codify the distances, but also for all the modules that work with these distances: adders, multiplexers, comparators...
2. We have to change the input/output ports since not enough switches or push buttons are available.

- We are going to evaluate the impact of the first modification.

Calculating the delay and area of the basic cell for larger matrixes

1. The maximum mutation distance for an NxN matrix is 2N (obtained when all the bases in the source are substituted by all the bases in the target).
2. The number of bits needed to codify it are:
   - N_B=ceiling($\log_2$(2N))
3. Assuming that the area and the delay of the basic cell increases a 20% for each additional bit, the formulas that estimate the area and the delay are:
   - Delay of the basic cell (N_B) = (15.5 + (N_B -4)*0.2*15.5) ns
   - Area of the basic cell (N_B) = (43 + (N_B -4)*0.2*43) LUTs

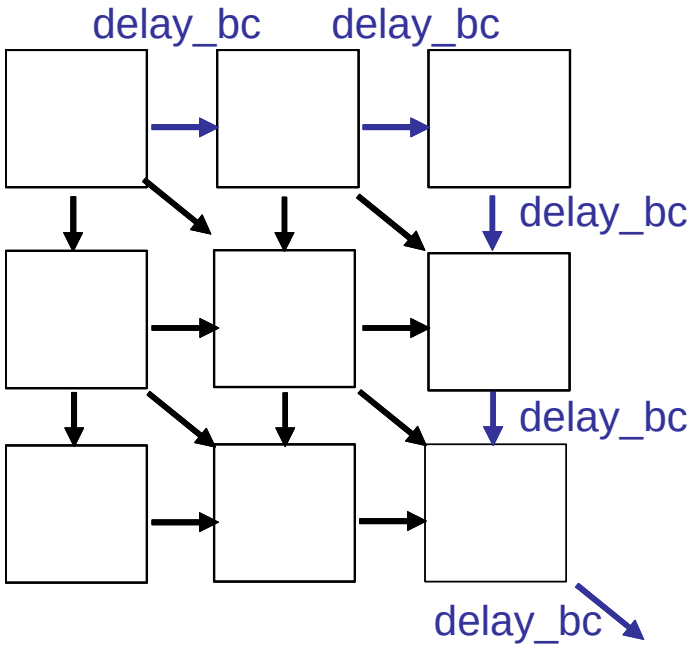
Delay of our initial basic cell: 15.5

Area of our initial basic cell: 43

Calculating the delay and area of a NxN matrix

- Delay of the NxN matrix (N) = $(2N - 1)^*$ $Delay_{bc}$ (N_B)
- Area of the NxN matrix (N) = $(N^2)^*$ $Area_{bc}$ (N_B)

Example for a 3x3 matrix



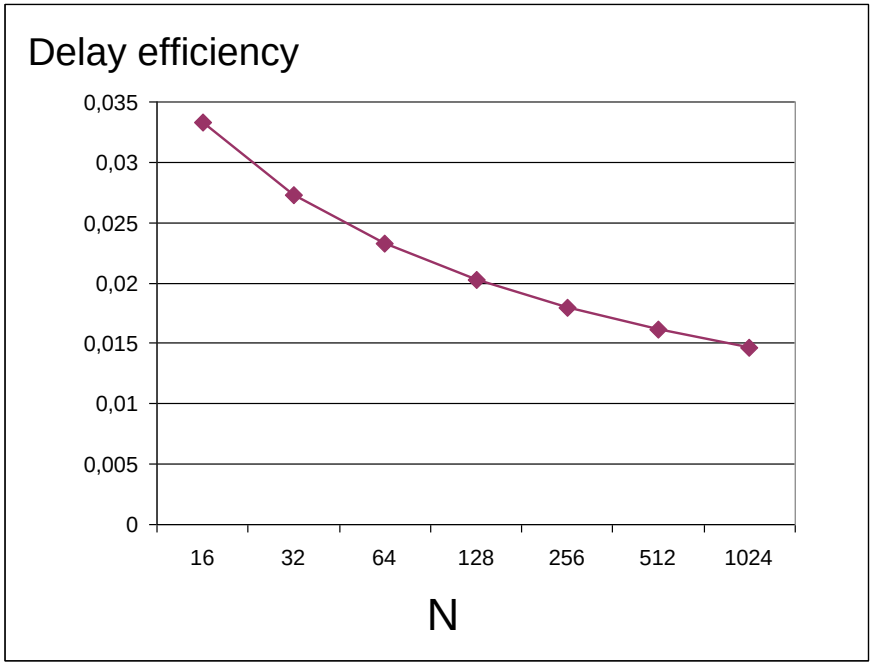
Area: 9 * $Area_{bc}$

Maximum delay: 5* $Delay_{bc}$

The efficiency of our design decreases with N

- Delay Efficiency of the NxN matrix (N) = N/ Delay of the NxN matrix (N)

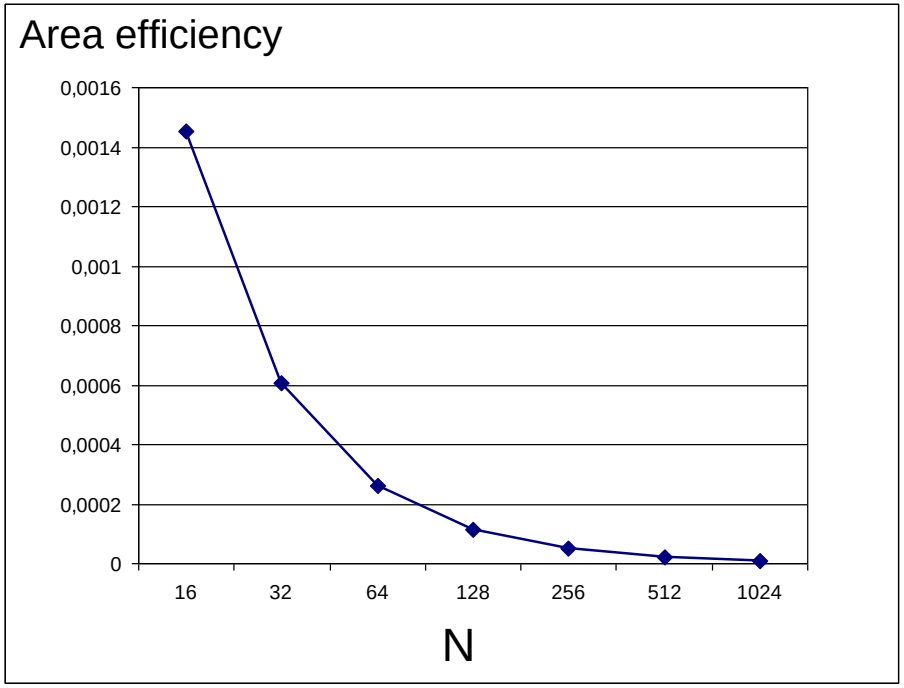
- The Delay Efficiency decreases more and less linearly.
- This is quite common for complex algorithms.
- The reason is that as N increases our algorithm demands more comparisons.
- If these delays are not acceptable we should try other comparison algorithms.



The efficiency of our design decreases with N

- Area Efficiency of the NxN matrix (N) = N/ Area of the NxN matrix (N)

- The Area Efficiency decreases more or less quadratically.
- This is much worst than in the previous case and it is the worst limitation of combinatorial design.
- However, it is also easier to solve:
  - **Instead of duplicating the resources, use twice the same hardware.**



- This can be done using **sequential elements** that store partial results and allows reusing the HW resources.

Index

- Digital logic Design for Bioinformatics
- Selecting the platform: why FPGAs?
- Outline of the design flow
- Evaluating our design
- **Conclusions**

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 have designed a circuit that implements the Smith and Waterman algorithm for DNA comparisons
- We have applied the following techniques:
  - Divide-and-conquer
  - Hierarchical design
  - Modular design

- With these techniques we have solved a problem that initially looked too complex.

Conclusions

- We have implemented a small system and tested it on an FPGA
- Finally, in this session we have evaluated our solution for larger systems using a theoretical model:
    - The Delay Efficiency decreases more or less linearly.
    - The Area Efficiency decreases more or less quadratically.
- To improve the Delay efficiency we may try other comparison algorithms.
- To improve the Area efficiency we should use sequential elements.

There is still some work needed in order to have a powerful implementation of the Smith and Waterman algorithm, but the techniques that we have learnt in this project will be the key for that design.