Faculty Project Description

Digital Logic for Medicine and Biology Research: A hardware implementation of the Smith-Waterman algorithm for DNA comparison

1. Overview

Bioinformatics is a major research field that directly influences both Biology and Medicine [1]. DNA is the main target of bioinformatics since it stores the genetic information that influences the most important characteristics of an organism. Dealing with DNA is not a simple issue. The main problem is that DNA is a huge data-structure. For instance, a human genome contains approximately 3 billion DNA base pairs. Hence, traditionally, only supercomputers could deal with it. However, recent research has proposed to develop hardware implementations of the available comparison algorithms using programmable hardware devices, such as FPGAs (Field-Programmable Gate Arrays**)** [2]. This option not only provides a high performance, but it is also very affordable.

2. Impact of the problem's solution on society

DNA consists of two long polymers of simple units called nucleotides. Each nucleotide contains a nucleobase, and the sequence of these bases is the key to store the genetic information. The DNA segments that carry this information are called genes, and the complete set of this information in an organism is called its genotype. A gene is a unit of heredity and is a region of DNA that influences a particular characteristic of an organism. Genes control the characteristics that an offspring will have by transmitting information in the sequence of nucleotides. The genetic information is coded in the DNA using only four different types of nucleobases, namely: adenine (A), cytosine (C), guanine (G) and thymine (T).

Since 1977, the DNA sequences of hundreds of organisms have been decoded and stored in databases. DNA comparisons are very useful both for Biology and Medicine. In Biology this information is used to analyze the evolution of the species. Thanks to the bioinformatics tools that compare DNA sequences, biologist can trace the evolution of families of organisms by measuring the changes in their DNA. With this analysis, biologists can better understand how the different species have evolved, and even identify which mutations led to new species.

In Medicine, DNA analysis is used for many different purposes. For instance it is very useful to find correlations between a given disease and DNA information, and if this correlation is strong, it can be used to identify who should start a preventive treatment. Another important application is the analysis of cancer mutations, which is a very active research field. In a cancer, the genomes of affected cells are rearranged in complex ways, and currently this process is frequently unpredictable. DNA comparisons can be used to track this process, and a better understanding of these mutations can be extremely useful to deal with this pathology. A similar approach can be used to better understand the evolution of viruses, identifying where the new viruses came from. This information can be very important when fighting against dangerous mutations of previous viruses.

All these important research activities demand high-speed DNA sequence comparisons, frequently comparing a given DNA sequence with the sequences stored in large scientific databases. However, these comparisons are tremendously computation intensive. As a result of this, traditionally they were only carried out using supercomputers. For instance Cray, one of the most important companies that develop supercomputers, includes in its systems a software application that compares DNA sequences using the Smith-Waterman algorithm. This demonstrates that bioinformatics has become a computational-hungry research area. Unfortunately supercomputers are very expensive, and not all the researchers can access this useful infrastructure. Moreover, typically supercomputers are shared among different research groups, and their use may become a bottleneck.

FPGAs (Field-Programmable Gate Arrays) are programmable hardware devices that can be customized to implement complex digital systems. In recent years FPGAs have exponentially increased their capacity and the supported clock frequency. For this reason, the research community is actively studying how to use FPGAs for high-performance computing. This approach has two main advantages. On the one hand, FPGAs are inexpensive when compared with supercomputers. On the other hand, they can even outperform supercomputers, since designers can develop application specific processors that take advantage of the inner parallelism of the given problem. Conventional processors are designed to provide good performance for any application, whereas application specific processors are just designed for a given problem, and all the hardware resources (datapath, memory hierarchy…) are optimized to achieve the maximum performance for that problem. In the case of the Smith-Waterman algorithm [3] several FPGA implementations have been proposed, achieving very important speedups when compared with traditional processors. In fact Cray has included FPGAs in some of its supercomputers and has extended its bioinformatics software in order to take advantage of them, obtaining a 26x speedup when compared with the same software executed in a Cray supercomputer without a FPGA [4]. An important disadvantage of this approach is that currently there is a lack of hardware designers. Hence, it is important to motivate students from the beginning in order to get that the future generations of computer engineers apply their knowledge in Science, Digital Logic Design and Computer Architecture to develop hardware solutions for some of our contemporary problems.

3. The project in the context of the CE-2004

This project has been designed as a part of a Digital Logic course, targeting basically Computer Engineering students. According to ACM/IEEE CE-2004 curriculum, computer engineers must possess the ability to design computer-based systems that include both hardware and software to solve novel engineering problems, subject to trade-offs involving a set of competing goals and constraints. In this project, students will focus on hardware design, and more precisely in Digital Logic, which is one of the basic areas identified in the CE-2004. In particular, this project addresses several of the basic units specified in this document:

- CE-DIG2 Combinational logic circuits [core]
- CE-DIG3 Modular design of combinational circuits [core]
- CE-DIG6 Digital systems design [core]:
    - Hierarchical, modular design of digital systems
    - Programmable logic devices (PLDs) and FPGAs
- CE-DIG7 Modeling and simulation [core]

This project attempts to settle the knowledge that the students have acquired in the Digital Logic lectures regarding topics CE-DIG2 and CE-DIG3. In addition, it will represent their first contact with a complete digital design flow. Hence, we will assume that the students have a basic knowledge of combinational design but they are neither familiar with FPGAs nor with the design and simulation environment.

4. The Smith-Waterman algorithm

The Smith-Waterman algorithm is an optimal method to compare DNA sequences that can also be used for a similar analysis of other structures such as Ribonucleic acid (RNA) molecules. Given two initial sequences represented as two strings, this algorithm compares segments of all possible lengths of these two strings measuring the mutation-distance between each pair of segments. In order to improve the efficiency of the process, the algorithm uses the results of the previous comparisons to evaluate the distance of the new ones. From a hardware perspective, this is a perfect example of a 2-dimensional iterative network. Basically the algorithm can be implemented as a matrix of cells where each cell compares two substrings and propagates its result to the other cells.

The mutation-distance between two sequences represents the mutations needed to go from one sequence (called Source or just S) to the other (called Target or just T). Three different mutations are considered: insertion, deletion and substitution, each of these mutations have a distance assigned. Typically the mutation distance for the insertion and deletion operations is 1, and for substitution is two. The reason is that a substitution produces the same result than the combination of an insertion and a deletion. Figure 2 present an example for these mutations.

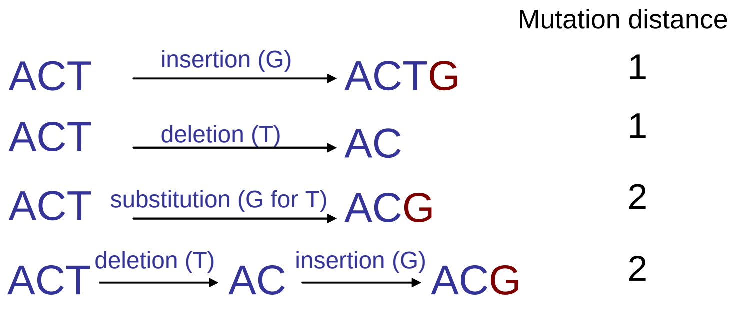

Figure 2. Example of mutations

Figure 3 depicts the basic structure of the implementation of the Smith-Waterman algorithm, and the functionality of the basic cell, including the inputs and outputs of cell (i,j), and the border values (the numbers that go from 0 to j+1 and from 0 to i+1 respectively). In addition, it presents the formula used to compute the distance (*d*) based on the previous intermediate results (*a*, *b* and *c*). Basically, each cell receives the distances a, b and c from the previous cells:

- a: distance to go from S_0-S_{i-1} to T_0-T_{j-1}
- b: distance to go from S_0-S_i to T_0- T_{j-1}
- c: distance to go from S_0-S_{i-1} to T_0- T_j

Then each cell updates these distances carrying out a mutation if needed to go from S_0-S_i to T_0-T_j :

- b + 1 represents the insertion of T_j
- c + 1 represents the deletion of S_i
- a + 2 represents the substitution of S_i for T_j. This is only needed when S_i and T_j are different (this is the reason for the *if* condition in the formula).

Finally the option with the smallest penalty is selected.

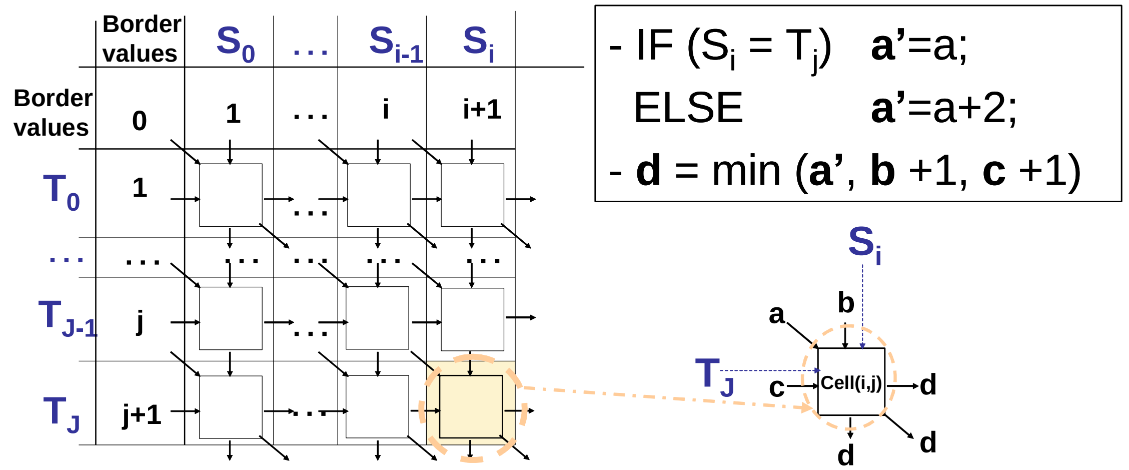

Figure 3. Matrix of basic cells for the Smith-Waterman algorithm, and description of the functionality of the basic cell.

Figure 4 includes a clarifying example where the source sequence "ACT" is compared with the target sequence "CGA". Figure 4.a presents the initial matrix including only the border values and the output of cell(0,0). Figure 4.b presents all the intermediate values, and the final result, which is the one obtained by the cell in the bottom-right corner. Finally Figure 4.c illustrates the meaning of the result obtained. In this case the distance is 4. That means that it is possible to go from the Source to the Target, for example, using four simple mutations (deletion or insertion), as depicted in the figure. Other options are also possible, but all of them would have a penalty of 4 or more.

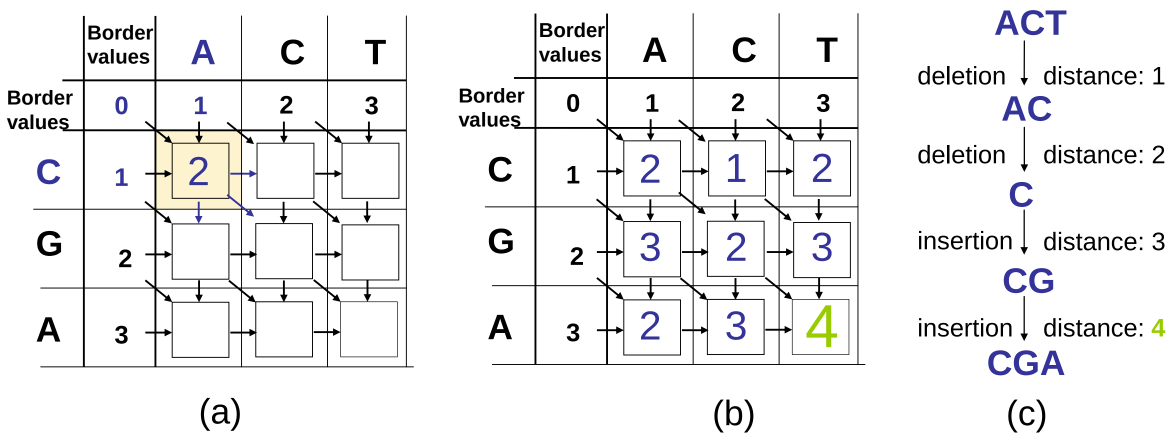

Figure 4. (a) Initial matrix with the output of cell(0,0). **(b)** Final matrix. **(c)** Steps to go from the Source sequence to the Target sequence.

One of the main advantages of this algorithm is that not only it computes the distance between two sequences, but if the intermediate results are analyzed, it can also be used to identify similar substrings. This is very useful to identify shared DNA patterns. However, the complexity of the algorithm is very high, because N*M comparisons are needed in order to calculate the distance between a sequence of N bases and a sequence of M bases. A hardware accelerator can be extremely helpful for this problem since it can take advantage of the inner parallelism of the algorithm.

Obtaining accurate measurements of the mutation distance is important not only in Medicine, where a small difference can be the key to identify a problem, but also in Biology, because the DNA of different species is frequently very similar. For instance, humans and chimpanzees share 96,1% of their DNA pattern. And in fact, only 1,2% of the DNA is different, the remaining 2,7% are replicated fragments of DNA that are present in both species, but different number of times.

5. Description of the project

The students in this project, working in groups of two people, will design a simple implementation of the Smith-Waterman algorithm that it is broadly used for DNA comparisons. The design is basically a matrix of cells, where each cell is a combinational circuit. They will start designing the basic cell of the Smith-Waterman algorithm, using a design environment based on schematics. Then, they will use the basic cell to build a 2x2 matrix that compares two sequences of 2 nucleobases and calculates the distances between them. They will learn how to simulate the design, and how to implement it and test it in the FPGA.

5.1 Resources needed

The material needed for this project consists just of a computer and a FPGA. Most computer engineering labs already have these elements. If FPGAs are not available, they can be ordered at a very affordable prize using the discounts for universities that most vendors offer. For instance, using the *Xilinx University program* [5], any faculty member can buy the *Basys System Board*, which includes a *Xilinx Spartan 3-E FPGA,* a simple extension board, for just 59$. For most digital design projects this system is just perfect. However, if the FPGA is also going to be used for more advanced courses, more powerful FPGAs are recommended. In the case of the Xilinx University program the *Virtex-II Pro Development System* ($299 academic) offers a good price/performance trade-off, whereas the *XUPV5-LX110T Development System* ($750 academic) offers the best performance. Regarding the software requirements, *Xilinx ISE® WebPACK™* Software includes all the software tools needed for this project and it is available free of charge [6]. Other vendors also offer free software tools and discounted FPGAs for universities. For instance Altera [7] offers the *Quartus II Web Edition* software that can be downloaded from its webpage at no cost.

If it is not possible to buy any FPGA, still 95% of the project can be done just using the FPGA software tools. The only limitation of not using FPGAs is that the students could not touch their designs. This is a pity because if we want to stress that logical design can be useful for real-world problems, it is important that the students realize that their designs are alive outside their simulation environment.

5.2 Learning objectives and trade-offs addressed

The following list encompasses the main learning objectives and trade-offs addressed in this project:

- FPGAs:
  - To understand the FPGA's architecture and its design flow
  - To identify the FPGA's pros and cons when compared with conventional processors and Application Specific Integrated Circuits.
- Combinational Hardware design-flow:
  - To apply a Divide-and-Conquer approach and Hierarchical design techniques in order solve complex problems.
  - To use Modular design techniques in order to reuse previous work and drastically speed up the design process.
- Design evaluation:
  - To learn how to simulate your design

- Project related topics:
    - To understand the potential impact of Bioinformatics in our society.
    - To identify the need for affordable high performance computing platforms, and learn how digital design and FPGAs can provide a solution for this need.

These topics will be addressed both in the lectures and in the labs. The idea is to illustrate some of the most important ideas and trade-offs present in digital-logic design. In the following paragraphs, I will explain some of them with a little bit more detail.

Students must understand why hardware design can provide better performance than conventional processors at the cost of a more complex design-flow. In addition they must understand that FPGAs are not the most efficient hardware implementation, since the majority of the area is wasted with unused resources. However, thanks to their regular structure and programmable nature, they can be used to implement almost any digital design, and the time-to-market will be much faster than when developing an Application Specific Integrated Circuit (ASIC). **Hence, processors, FPGAs and ASICs provide different trade-offs between efficiency, design-effort and development cost.**

Another essential idea that students must assimilate is the importance of **modular and hierarchical design**. Conventional design techniques based on truth tables and Karnaugh maps are only useful for very small designs. When dealing with a large problem, the first thing that the designer must do is to apply a **divide-and-conquer approach** that identifies several smaller sub-problems that the designer can solve. Iterative networks are a great example of this idea, since they can be used to deal with large problems developing only a very small basic cell. The importance of modular design must also be pointed out. It makes no sense to design the same blocks every time you need them. **Reuse is the key for an efficient design-flow**. Those blocks that are frequently used must be included in a library in order to reuse them in the future.

One of the most important ideas that students must learn is that a design is not finished until it has been exhaustively simulated and tested. Students must always develop a test-bench for their designs. This test-bench must be used to carry out Functional simulations that will identify most errors in the design before downloading the design in the FPGA.

5.3 Project structure

This project has been divided into four units.

Unit 0: Lecture sessions. (3 hours). This project should be the first contact with FPGAs and a hardware design flow. Hence, the first two hours will be used to explain them, focusing on the trade-offs previously described. The last hour will be used to introduce the project.

Unit 1: Developing the basic cell (6 hours). Students must design a basic cell that implements the formula depicted in Figure 2.b. However, they cannot face this problem using only truth tables and Karnaugh maps, because there are too many inputs. Hence they must follow a modular design approach, identifying smaller sub-problems, which frequently can be implemented using standard modules available in the schematic library, and compose the global solution using these modules. In this case, they need a comparator, a 2:1 multiplexer, an adder (that will be instantiated three times), and a module that selects the minimum value among the three candidates (*min* module). The latter module is not available in the schematic library, but it is available in the solution packet. The students must start the project analyzing this module, simulating it and finally generating a schematic symbol in order to include it in the library and use it in the following steps. Once the students have identified all the modules needed for the basic cell, they will compose it with all these elements. They must write down their design on paper, and after that they will translate it into a schematic using the schematic editor, assuming that the intermediate values (a, b, c and d) are encoded using four bits. Then, they will design a test bench and simulate it.

Finally they will implement the design (they just need to select the implement design option in the menu).

Unit 2: Developing a 2x2 matrix using the basic cell (2 hours). As an example of a hierarchical design, the students will design a 2x2 matrix using the module developed in the previous unit (that will be instantiated four times). This design will compare two sequences of two bases. Then, the students will develop a test-bench again and they will simulate it using the functional simulation. Eventually, they will download and test the design in the FPGA. Since this project is the first contact of the student with FPGAs, I/O will be mapped to the simplest resources: switches, push-buttons and leds. Using these resources is straightforward, as the students only have to map the ports of their design to the proper FPGA pins (no drivers or interfaces are needed).

6. Adapting the project

The previous section describes the basic project. Whatever, if more time is available some interesting extensions can be added (these extensions are further described in Appendix B):

- **Designing the min module** (2 hours): instead of given the min module to the students, we can ask them to design it. The main interest of this step is that they will have to use both modular design (using multiplexers and comparators) and custom-logic design (using truth tables and Karnaugh maps).
- **Evaluating the design of a NxN matrix** (2 hours): in this extension the students must identify which modifications are needed in order to develop a NxN matrix. Basically, the number of bits used to encode the intermediate values must be adjusted according to the size of the matrix, and the size of some of the basic modules used in the design must also be adjusted. In addition, the students will carry out a theoretical analysis in order to estimate the delay, hardware cost and efficiency for a NxN design.
- **Post-Place & Route simulations** (1 hour): functional simulation is useful because it is fast and it can be done even before synthesizing the project. However, the results are not totally accurate because it does not take into account hardware delays. These delays are accurately included in the Post-Place & Route simulation at the cost of a slower simulation speed. **Students must compare both simulations, identify the differences, and explain them**. We are not looking for a detailed explanation, but we just want them to understand that in a real circuit everything has a delay, and for that reason the output of a digital circuit frequently generates wrong values until it becomes stable.

Not using FPGAs: using FPGAs to implement a combinational circuit is quite straightforward. However, if FPGAs are not available, or if the available time is very restricted, we can just leave out the last step in Unit 2 and use only the simulation environment to simulate the designs.

7. Description of the additional material.

The following files are available for this project:

1. A background lecture
2. A student project assignment
3. A faculty project description
4. A summary lecture
5. A solutions packet that includes the following files:

a. A project report solution
b. min.sch: schematic of the min module for the ISE environment. **This file must be delivered to the students unless the instructor decides to extend Unit 1 as suggested in the Appendix B.**
c. basic_cell.sch: schematic of the basic cell module for the ISE environment
d. matrix.sch: schematic of the 2x2 matrix for the ISE environment
e. matrix.ucf: constraint file for the XUP Virtex-II Pro Development System
f. SW directory: and example of a Xilinx ISE 9.1 project including all the files.

6. partial_min.sch: schematic of the min module for the ISE environment but without the custom logic. **This file must be delivered to the students if the instructor decides to extend Unit 1 as suggested in the Appendix B.**
7. Appendix A: Introduction to *Xilinx ISE*: Step-by-step examples describing how to work with the schematic editor of Xilinx ISE 9.1.
8. Appendix B: description of the project extensions.

References

[1] www.bioinformatics.org
[2] Storaasli, Olaf, Yu, Weikuan, Strenski, Dave, and Malby, Jim; Performance Evaluation of FPGA-Based Biological Applications. Cray Users Group Proceedings, Seattle WA, May 2007
[3] T. F. Smith, M. S. Waterman, Identification of common molecular subsequences, Journal of Molecular Biology, Volume 147, Issue 1, 25 March 1981, Pages 195-197,
[4] Margerm, Steve, and Maltby, Jim; Accelerating the Smith-Waterman Algorithm on the Cray XD1, Cray WP-0060406 2006.
[5] http://www.xilinx.com/univ/
[6] http://www.xilinx.com/tools/webpack.htm
[7] http://www.altera.com/education/univ/unv-index.html