TÉCNICO LISBOA

FPGA-Based Accelerator for High-Order Epistasis Detection

Gaspar Minderico Ribeiro

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Doctor Aleksandar Ilic
Doctor Nuno Filipe Simões Santos Moraes Neves

Examination Committee

Chairperson: Doctor Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Doctor Aleksandar Ilic
Member of the Committee: Doctor Ricardo Jorge Fernandes Chaves

January 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to express my gratitude to my supervisors and to everyone that helped me during the entire development of this thesis, namely Dr Aleksandar Ilic, Dr Sergio Santander-Jiménez for their help and patience week after week, and especially to Dr Nuno Neves for his incredible availability throughout all stages of development. I would also like to thank IST and INESC-ID for providing me with the knowledge and tools needed for the development of this project.

I also want to acknowledge the continuous support that my family has provided me with, not only during the development of this thesis but throughout my entire life.

At last, a big thank you to all my friends, who thanks to their support and friendship and all the good moments that we shared, made the years that I spent in this institution truly unforgettable.

Abstract

The classic approach to Genome-Wide Association Studies (GWAS) attempts to find a relation between one or more Single Nucleotide Polymorphisms (SNPs) contributing to the manifestation of a certain disease or trait. However, most genetic diseases do not depend on the individual effect of one or more SNPs, but rather on the interactions between several SNPs, also known as epistasis. Detecting epistasis for high-order interactions results in a huge computational complexity, as the number of SNP combinations evaluated exponentially grows with the order of the interactions. For this reason, state-of-the-art exhaustive search-based methods for epistasis detection rely on Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) to provide high-performance solutions for second-order interactions. Nevertheless, only rare attempts are made to tackle third-order interactions, and the ones that exist are not scalable for higher-order interactions and could be further optimized.

In this thesis, the flexibility and scalability limitations of the existing FPGA implementations are addressed, while a novel parameterizable architecture is proposed that enables the deployment of FPGA-based accelerators targeting any order of interactions in modern FPGAs. The resulting implementations can target any dataset size and provide higher performance and energy efficiency than the state-of-the-art FPGA-based architectures. In particular, performance gains of up to 80x and 3x are achieved when compared to existing second and third-order implementations, respectively, while also providing average energy efficiency improvements of 79x. Finally, the proposed architecture allowed for the implementation of a fourth-order epistasis detection accelerator in an FPGA platform.

Keywords: Genome-Wide Association Studies, Epistasis Detection, FPGA Accelerator, Single Nucleotide Polymorphism, Domain-Specific Architectures

Resumo

A abordagem clássica aos estudos de associação do genoma completo (GWAS) procura encontrar uma relação entre um ou mais polimorfismo de nucleotídeo único (SNP) contribuírem para manifestação de uma determinada doença ou característica física. No entanto, a maioria das doenças genéricas não dependem dos efeitos individuais de um ou mais SNPs, mas sim da interação entre vários, ao que se dá o nome de epistasia. A deteção de epistasia para interações de ordem elevada resulta numa elevada complexidade computacional, visto que o número de combinações de SNPs a avaliar aumenta exponencialmente com a ordem das interações. Por esta razão, os mais recentes métodos exaustivos para a deteção epistasia são implementados em GPUs e em plataformas FPGA para a obtenção de soluções de alto desempenho para interações de segunda e terceira ordem. No entanto, poucas implementações em FPGAs têm sido desenvolvidas para detetar interações de terceira ordem, e aquelas que existem não são escaláveis para ordens de interações mais elevadas e têm ainda espaço para otimizações ao nível da arquitetura.

Esta tese aborda as limitações de escalabilidade e flexibilidade das implementações existentes em FPGA ao propor uma nova arquitetura genérica para qualquer ordem de interações. As implementações resultantes são mais rápidas e eficientes do que as existentes, resultando em ganhos de desempenho de 3x, quando comparados com implementações terceira ordem, e melhorias de eficiência energética de, em média, 79x. Por fim, a arquitetura proposta permitiu a implementação de um acelerador de deteção de epistasia de quarta ordem numa plataforma FPGA.

Palavras-chave: Estudo de Associação do Genoma Completo, Deteção de Epistasia, FPGA, Polimorfismo de Nucleotídeo Único, Arquiteturas de Domínio Especifico, Aceleradores baseados em FPGA

Contents

List of Tables

List of Figures

List of Acronyms

AMBA Advanced Multicontroller Bus Architecture.

ASIC Application Specific Integrated Circuit.

AXI Advanced eXtensible Interface.

BRAM Block Random Access Memory.

CPU Central Processing Unit.

CTU Contingency Table Unit.

DMA Direct Memory Access.

DNA Deoxyribonucleic Acid.

DSA Domain-Specific Architecture.

DSP Digital Signal Processor.

ECC Error Correcting-Code.

EDP Energy-Delay Product.

FPGA Field Programmable Gate Array.

GPP General-Purpose Processor.

GPU Graphical Processing Unit.

GWAS genome-wide association study.

HP High-Performance.

IP Intellectual property.

LUT Look-Up Table.

MIU Mutual Information Unit.

PE Processing Element.

PL Programmable Logic.

PopCount Population Count.

PS Processing System.

SNP single-nucleotide polymorphism.

SoC System on Chip.

Chapter 1

Introduction

During the past two decades, Genome-Wide Association Studies (GWAS) have been used to find a relationship between genetic variations and the manifestation of a phenotype, such as a disease or trait [1]. Typically, GWAS attempt to find an association between one or more single-nucleotide polymorphism (SNP) (which is the most common genetic variation in the human genome), with a certain disease. This association is usually performed via an observational case-control study, by comparing several SNPs from a group of patients that suffer from the disease (cases), with a group of people that do not suffer from the disease (controls). This approach has been used with variable degree of success, in identifying genetic markers associated with some diseases, such as age-related macular degeneration [2]. However, some more complex genetic diseases, such as type II diabetes [3], are not possible to explain using the traditional GWAS approach. This happens largely because gene-gene interactions and environmental factors are ignored when following a classic GWAS approach [4]. The effect of the interaction between multiple SNPs in the manifestation of a certain phenotype is known as epistasis [5].

Epistasis detection plays an important role in understanding complex traits in humans. In particular, it has been successfully used to study several complex diseases such as Late-Onset Alzheimer's Disease [6], asthma [7] or type II diabetes [8]. However, the detection of pair-wise epistatic interactions in the large datasets, which can contain hundreds of thousands of SNPs, is a highly computationally intensive task, as the number of SNP-SNP combinations to be evaluated increases exponentially with the number of SNPs in the targeted dataset. The detection of higher-order epistatic interactions is even more challenging since the number of combinations to be tested also increases exponentially with the order of the interactions. For example, almost 200 million pair-wise combinations exist in a dataset with 20 000 SNPs, but the same dataset contains more than 1.3 trillion and 6.6 quadrillion third and fourth-order combinations, respectively.

Due to the exponential increase of the number of combinations to be evaluated with the order of the interactions, most exhaustive-based methods for detecting epistasis (in which all the possible combinations are individually evaluated), are typically limited to second-order interactions, since performing higher-order epistasis detection over a larger dataset often results in an infeasible execution time. In addition to methods based an exhaustive search, other methods attempt to reduce the computational

burden by reducing the search space [9], therefore only evaluating a fraction of the SNPs available in the dataset. However, as the pre-filtering stage might remove some potentialy relevant SNPs to the phenotype, those methods are generally less accurate than exhaustive search across all possible combinations.

1.1 Motivation

Due to the computational complexity of performing epistasis detection based on exhaustive search, most implementations only target second-order or, in some cases, third-order interactions. However, detecting higher-order epistatic interactions for complex traits is absolutely crucial as they are likely to have severe implications on certain phenotypes [10]. As such, the need arises for better algorithms, that take advantage of the technological advances in the hardware field, to overcome the computation burden associated with performing higher-order epistasis detection based on an exhaustive search over large datasets.

The first proposed methods to detect epistatic interactions based on exhaustive search methods relied on software solutions running on multi-core Central Processing Units (CPUs) [11]. However, due to their limitations in exploiting the existent data parallelism, CPU based implementations are typically inefficient in detection of epistatic interactions. For example, processing a dataset with 500 000 SNPs on 2 quad-core Intel Xeon processors takes 19 hours for second-order epistasis [12]. Such a high execution time indicates that it is unfeasible to rely on a traditional CPU based approach to preform high-order epistasis detection. Therefore different hardware platforms that can efficiently take advantage of parallelism existent in epistasis detection are needed.

One of the tendencies to overcome the limitations of general-purpose processors is the use of domain-specific architectures (DSA) to execute part of an application [13]. As DSAs are made specifically to deliver better performance and energy efficiency in a class of applications, such as deep learning, where neural network processors are much more efficient than a general-purpose processor [14]. As they generally do not execute an entire application, DSAs are often referred to as accelerators.

Graphics Processing Units (GPUs) are one of the most well known and widely used DSAs, thanks to their high core count, GPUs are efficient in exploring the data-level parallelism, as multiple operations can be executed in parallel [15]. Contrarily to general-purpose processors, GPUs have been proven effective to perform second-order [16], and even third-order in over relatively big datasets in a reasonable amount of time [17], by taking advantage of their high core count to process several combinations in parallel. However, even with the very efficient implementations that already exist for second and third-order exhaustive search epistasis detection on GPUs [16], not many attempts to performing fourth-order epistasis detection based on exhaustive search have been made, due to the massive amount of combinations to be tested even for a dataset with just a few thousand SNPs.

Field Programmable Gate Arrays (FPGAs) are a type of DSAs that can be reconfigured to allow for the implementation of any logic circuit. Due to their flexibility, FPGAs are used as accelerators for a wide range of applications, including bioinformatics [18] [19] [20], where a logic circuit that exploits the data

parallelism existent in bioinformatics applications, such as in epistasis detection.

Despite the added difficulties that arise from FPGA design, some successful attempts at deploying custom architectures to perform second [21] and third-order [22] exhaustive epistasis detection in FPGA have been proposed. Although they offer higher performance and energy efficiency CPU based approaches, the state-of-the-art solutions are not easily scalable and only work for a limited range of patients. Improvements to the FPGA technology that have been made in the last decades [23], suggest that, in the future, higher-order exhaustive search epistasis detection could be performed in FPGAs. However, due to the lack of scalability with the order of interactions of the existent implementations and complexity of hardware design, the need for a novel method to deploy efficient custom architectures to perform higher-order epistasis detection in FPGAs arises.

1.2 Objectives

The work presented in this thesis was developed under the HiPERBio project, being developed in INESC-ID, which aims to implement high-performance and energy-efficient processing of bioinformatics applications, by exploiting the capabilities of emergent heterogeneous systems. Accordingly, the work presented in this thesis falls in the scope of the HiPERBio project, as it explores the use of FPGAs to implement a custom accelerator for epistasis detection.

As suggested in [21] and [22], FPGAs can be used to support specialized architectures for high-order epistasis detection that provide solid performance with lower power consumption. Nevertheless, there is still an opportunity to further explore this technology, by creating a generalized method that allows for the easy deployment of new accelerator architectures to perform exhaustive search epistasis detection for any order and over any dataset, implementable in any FPGA and scalabe with its resources.

Accordingly, the main objective of this master thesis is to address the main limitations of state-of-the-art implementations and propose a novel method to generate accelerator architectures to perform K-order epistasis detection that are implementable in modern FPGAs. The proposed method aims to provide custom architectures that outperform the existing second and third-order implementations.

1.3 Contributions

A thorough study on the implementation of a K-order epistasis detection accelerator in FPGA platforms, identified the main limitations of the state-of-the-art solutions. Furthermore, the contributions of this master thesis include new FPGA-based accelerators for epistasis detection based on exhaustive search, that tackles the main limitations of the state-of-the-art implementations, by providing more flexible designs and higher performance and energy efficiency. The proposed architecture enables the implementation of FPGA-based accelerators targeting epistasis detection of any order using datasets with any number of patients ans SNPs.

Specifically, the main contributions made by this thesis are the following:

- A novel approach to generate specialized architectures to perform epistasis detection of any order and targeting any number of patients in an efficient way, that can be implemented in any FPGA, and scalable with the number of resources available, which means that the more available FPGA resources, and the higher the clock frequency is, the better the performance of the implemented architectures is.
- Novel second and third-order architectures, based on the proposed method, that are more efficient than the state-of-the-art FPGA implementations, capable of achieving 3x speedup when using similar hardware, and an average of 79x better energy efficiency when compared against the existent second-order implementations.
- A new custom hardware architecture targeting fourth-order epistasis detection based on an exhaustive search, representing the first attempt at the deployment of a fourth-order implantation in FPGA platforms.

The proposed architectures were implemented in a ZYNQ-7 Mini-ITX and in a Xilinx Zynq UltraScale+ ZCU102 boards, where the FPGA-based accelerator runs at a clock frequency of 250MHz and 322MHz, respectively. The implementations targeting second order epistasis provide an energy consumption about 80x lower than the state-of-the-art implementation while achieving a similar execution time. The implementations targeting third-order interactions are also implemented in a Virtex-7 690T FPGA for better comparison against the state-of-the-art implementation, where a speedup of 3x is achieved when running at same clock frequency of 250MHz as the state-of-the-art FPGA-based accelerator.

1.4 Thesis Outline

This thesis is structured as follows. Chapter 2 provides background on the fundamental notions of genetics needed to understand the concept of epistasis, as well as a summary regarding the different methods used for epistasis detection, with particular emphasis on the state-of-the-art exhaustive search implementations on FPGAs. Chapter 3 details the proposed method for the generation of special architectures for epistasis detection, maintaining the generality in regards to the order of the interactions and the number of patients to be targeted, and the bandwidth used between the memory and the FPGA. Chapter 4 refers to the implementation of the architectures generated using the proposed method, targeting different datasets and different orders of interactions ranging from the second to the fourth-order. This chapter also provides details regarding the boards that are used to implement said architectures. The results obtained from implementing some the architectures detailed in the previous chapter, as well as a comparison between the obtained results and the existent state of the art comparable FPGA implementations are presented in Chapter 5. At last, Chapter 6 states the conclusions taken from the proposed works and proposes future work on the topic.

Chapter 2

Background: Genetic Studies and Epistasis Detection

Thanks to the recent technological advances in the genetics field, there has been a rapid growth of the available genetic data in the past 20 years [20]. As such, the need to compute large amounts of data has increased. Bioinformatics applications require the processing of increasingly large datasets, resulting in a tremendous computational load to extract relevant information from the available data. GWAS [1] are commonly used to identify the likeliness of an individual to display a certain phenotype, which is a set of observable characteristics such as a disease or a physical trait, by studying their genetic material, particularly the SNPs which are the most common types of mutations in the human genome. The most common approach to GWAS consists of a classic case-control study, where the SNPs of a group of individuals who are diagnosed with a specific disease (cases) are compared with a group that are not diagnosed with the disease (controls).

Epistasis detection is a particular approach used in GWAS that has only recently started to be used to identify genetic interactions that may be responsible for certain diseases. Instead of focusing on the effects of individual SNPs, epistasis detection [5, 24] aims to uncover meaningful interactions between two or more SNPs for the identification of more complex diseases. Consequently, the amount of processing power required is much larger, due to the number of combinations that needs to be tested, which increases exponentially with the order of interactions, and with the number of SNPs available in the used dataset. As such, performing high-order epistasis detection on a dataset containing hundreds of thousands of SNPs is currently a very time-consuming process, even in the high-end CPUs [25]. As an example, a machine with two Intel Xeon E5-2667v4 eight-core CPUs @ 3.2 GHz and 256 GB of RAM took more than 5 days to detect second-order epistasis using PLINK [26] on a dataset with 130,052 SNPs and 48,726 samples. In fact, detecting epistasis of higher order is often infeasible with the methods available today, due to the complexity of the problem and the size of the datasets.

This chapter provides detailed insights regarding the problem of epistasis by presenting a background on the underlying genetics concepts, a review of the most relevant implementations proposed in the scientific community, and a discussion on the recently proposed second and third-order implementations

of epistasis detection on FPGA-based platforms.

2.1 Fundamentals on Genetics

In order to understand epistasis, it is fundamental to realize the basis of genetics and heredity. All the genetic information of a living being is encoded in the Deoxyribonucleic acid (DNA). The DNA is a molecule composed of two intertwined polynucleotide chains creating a double helix. Each polynucleotide chain is formed by a group of nucleotides, linked together by covalent bonds. The nucleotides themselves are the basic constituents of the DNA molecule, and are composed of a sugar, called deoxyribose, a phosphate group, and one of four nitrogen nucleobases: Adenine (A), Cytosine (C), Thymine (T), and Guanine (G). It is through the nitrogen bases in each nucleotide that the two DNA strands are linked, forming a base-pair, with the combinations A-T and C-G.

Genes, defined in biology as sequences of nucleotides, are responsible for the transfer of information between an individual and their offspring, causing the inheritance of physical traits (such as the color of the eyes or hair). The whole human genome is constituted by 23 pairs of chromosomes (sequences of DNA), in which, each gene occupies a specific fixed position inside a chromosome, named a *locus*. A variant of a gene in the same *locus* is called an allele (different alleles can cause different phenotypical traits). Each *locus* is defined by two alleles, one in each chromosome of the pair, each one of those alleles is inherited by one of the parents. A *locus* is called homozygous when both alleles are the same, and heterozygous if they are different.

Table 2.1 is called a Punnett square, which is used to represent the probability of an offspring to display a certain phenotype, but is also useful to demonstrate the concept of dominant and recessive allele. As observable in the Punnet square 2.1, when a *locus* is heterozygous, the observable phenotype is asserted by the one of the alleles (A in this case), making it the dominant allele. Therefore, a recessive phenotype will only manifest itself if both alleles of an individual are of the recessive type (a in this case). The two alleles that compose a *locus* are inherited from the parents (each parent contributing with one alleles).

When a *locus* is heterozygous, the observed phenotype will be asserted by the dominant allele. Table 2.1 exemplifies the concept of dominance, as it shows that certain recessive phenotype, will only manifest itself if both alleles of an individual are of the recessive type, where those alleles are inherited one from the parents, the manifestation of the physical trait is represented as a 1 in the table.

Table 2.1: Punnet square example

	Father	
Mother	A	a
A	0	0
a	0	1

2.1.1 Genome-Wide Association Studies

A Genome-wide association study (GWAS) is an observational study of genetic variants in the entire genome of an organism, encompassing multiple individuals, displaying different phenotypes for a certain trait or disease. The main objective of such methods is to find if any genetic variations can be associated with the genetic trait being tested.

Single-nucleotide polymorphism (SNP) is one of the most common genetic variations used in GWAS. They represent a difference in a single nucleotide, at a specific position in the DNA. As an example, for a specific DNA position, where in most individuals the nucleotide cytosine (C) appears, there is a small group of people that instead have the nucleotide adenine (A) in that same position. In fact, SNPs are estimated to occur, on average, in 0.1% of nucleotides, which translates about 10 million SNPs in the human genome [27]. Several SNPs have been linked to the susceptibility to some diseases and traits [28] such as myocardial infarction [29] or age-related macular degeneration [30]. SNPs may also help to predict the susceptibility to environmental factors and the response to certain drugs. However, most SNPs do not have any documented effect on the individual. One SNP can assume 3 different states, *Homozygous major* (when both alleles are of the dominant type), *Heterozygous* (when the alleles are different) and *Homozygous minor* (when both alleles are of the recessive type), that are encoded according to Table 2.2.

Table 2.2: SNP Coding

Alleles	Genotype	Symbol
A/A	Homozygous major	0
A/a a/A	Heterozygous	1
a/a	Homozygous minor	2

GWAS typically focus on the association between SNPs and the phenotype, considering that the SNPs have independent effects on the said phenotype or disease. GWAS usually rely on performing case-control studies, using arrays of SNPs from individuals suffering from the disease (cases), and from individuals without the disease (controls). Usually, the GWAS attempts to identify which SNPs are responsible for the occurrence of the disease by comparing the allelic frequencies (frequency on an allele ate a particular *locus*) between cases and controls. In a traditional approach, each SNP is individually tested to find its statistical relation with a certain disease. This strategy provides limited results as it neglects the effect that the interaction of two or more SNPs (epistasis) may have on predicting more complex diseases.

An example of a dataset used for GWAS is depicted in Table 2.3, where the lines represent different SNPs, the columns represent the patients (where some are cases and the others controls), and the entries of the dataset represent the type of SNP for each patient, following the notation introduced in Table 2.2.

Table 2.3: GWAS dataset example

	0	1	2	3	...	N-1
SNP A	1	0	1	1	...	2
SNP B	0	1	2	1	...	0
SNP C	2	0	0	2	...	1
SNP D	0	2	1	0	...	1

2.1.2 Epistasis

Epistasis can be defined as the interaction between genes to the definition of a phenotype. It takes into account that a phenotype can be caused by not only the independent effect of a *locus*, but also by the effect that different *loci*, have on each other on creating that phenotype [24]. The term "Epistasis" was first used by William Bateson to describe the effect that an allele at one *locus* has on masking the effect of another *locus*, similarly to the concept of dominance but on an inter-*loci* level. Generally, epistasis exists when the effect of an SNP depends, or not, on the existence of another SNP.

Table 2.4 shows an example of two *loci* interacting epistatically according to Bateson's definition of epistasis [31], where 1 and 0 represent the manifestation (or not) of a certain trait, respectively. In the presented example, *loci* X is epistatic to *loci* Y, particularly, allele A in *loci* X is epistatic to allele B in *loci* Y, as it effectively masks its effect.

Table 2.4: Two *loci* interacting epistatically

	Loci Y		
Loci X	b/b	b/B	B/B
a/a	0	1	1
a/A	0	0	0
A/A	0	0	0

The definition of epistasis proposed by Bateson is also referred to as biological epistasis, as opposed to the statistical epistasis definition proposed in 1918 by R.A. Fisher [32], which is a departure from a linear model to describe the effect of alleles at different loci, regarding their contribution to a quantitative phenotype. The objective of the latter is to statistically classify the interactions found to discover their relevance in explaining a certain phenotype at a population level [9]. When using computational methods to detect epistasis, statistical epistasis is the relevant definition.

2.2 Epistasis Detection

Given its complexity, several difficulties arise from detecting epistasis. The enormous amount of combinations to be tested, given by:

$$\frac{\prod_{i=0}^{k-1}(N-i)}{k!}, \tag{2.1}$$

where N is the number of SNPs in the dataset and K represents the order of interactions, results in a huge computational burden when detecting epistasis in a large dataset and specialty when the order of interactions increases. As when the value of K increases, the number of tests to be made increases exponentially. Which is why the detection high order epistasis, where $K > 2$, on large datasets is very computationally demanding and often time-wise infeasible with the current technology. Also, Choosing the best statistical test (objective function) to evaluate each SNP combination is not straightforward, as there is no consensus about the best objective function and the statistical results still need to be biologically interpreted.

Different methods to detect epistasis attempt to overcome the high computational burden using different strategies. Those methods can be broken down into three main categories [9]. Exhaustive search methods (such as the ones employed in PLINK or BOOST [26] [11]) are considered the most accurate ones, as all possible combinations are evaluated independently. However, since the number of combinations increases dramatically with the order of interaction that is being evaluated and with the size of the dataset, so does the computational burden associated with testing all possible combinations, As such, most exhaustive search algorithms only target second-order interactions and are difficult to scale to higher orders.

A common alternative to exhaustive search is to pre-filter the available SNPs in an attempt to reduce the computational burden by diminishing the number of SNPs to test [33] [34]. To do so, the available SNPs are individually evaluated and filtered, reducing the number of SNPs that are used to perform epistasis detection to a small fraction of the original dataset. A shortcoming of this type of approach is a potential loss of accuracy, as only a small amount of combinations are effectively tested for epistasis. Non-exhaustive methods enhanced by artificial intelligence, such as random forests [35], Bayesian networks [36] and ant colony optimization [37], provide an alternative to the commonly used parametric statistical methods for detecting epistasis

However, the categorization of the methods is not consensual between authors, as a lot of different algorithms have been developed and several of them rely on approaches with two or more stages, therefore, fitting more than one category at the same time. Also, the landscape is in constant change due to recent developments, such as new or improved algorithms or the use of new hardware [21] or combinations of hardware in heterogeneous systems [25].

This thesis proposes an FPGA-based exhaustive search approach, for the detection of high-order epistasis, that tackles the problem of the computational burden by providing a method to generate highly optimized architectures for any order of interactions and dataset size. The method is based on the use of simple bit-wise operations to create contingency tables for all possible combinations, which are then processed according to an objective function so that meaningful information can be extracted from the contingency table.

2.2.1 Contingency Tables

Contingency tables reflect the frequency distribution of all possible SNP combinations in the dataset throughout the individuals. One contingency table is created for each of the existent unique combinations (given by Equation 2.1), which is dependent on the number of SNPs in the dataset and the order of the interactions. Since one SNP can assume 3 different states, the number of possibilities to evaluate within a combination depends on the order of the combination, and is given by 3^K. Each table has information referring to the cases and the controls separately, consequently, the number of entries in a contingency table is equal to $2 * 3^K$, hence, as the order of the combination increases, not only does the number of contingency tables to be created increases, but so does the number of possibilities to be tested to create each one.

Table 2.5 shows an example of a contingency table for a second-order interaction between SNPs. The columns represent the state of both SNPs that constitute the combination (following the notation introduced in Table 2.2), and each entry of the table, n_{xy}, denotes the number of times which that interaction occurs throughout all the individuals in the dataset.

Table 2.5: Contingency table for a pair-wise interactions

	0,0	0,1	0,2	1,0	1,1	1,2	2,0	2,1	2,2
Cases	n_{00}	n_{01}	n_{02}	n_{10}	n_{11}	n_{12}	n_{20}	n_{21}	n_{22}
Controls	n_{00}	n_{01}	n_{02}	n_{10}	n_{11}	n_{12}	n_{20}	n_{21}	n_{22}

2.2.2 Binary Representation

In an attempt to decrease the computation burden of epistasis detection, a new approach [11] proposed that the SNPs can be represented in a notation that allows for the use of simple bit-wise operations to calculate the entries of the contingency tables. This is done by coding an SNP as a combination of 3 bits in a one-hot notation, where the number of the bit that is high corresponds to the SNP type (as per Table 2.2). Using this representation, each SNP in a dataset is defined by three binary vectors, with as many entries as the number of patients. As such, each dataset contains three times the number of SNPs binary vectors with as many entries as the number of patients.

Table 2.6 shows one SNP, SNP Z, coded using a binary representation, resulting in the creation of 3 vectors, Z0, Z1, and Z2. Those vectors will be referred to as genotype zero vector (G0), genotype one vector (G1), genotype two vector (G2) of the SNP Z, respectively, throughout this thesis. When using a binary representation, 3 bits are needed to encode a single SNP. However, note that G2 can be inferred by a bit-wise NOR between G0 and the G1. This allows for the reduction of the size of each SNP to 2 bits. Therefore the dataset is only required to contain a number of binary vectors equal to the double of the number of SNPs, with as many entries as patients.

With such a convenient representation, to generate the entries of each contingency table, it is only necessary to perform a bit-wise AND between all the three vectors of the K SNPs that compose the K^{th} order combination. Note that when an AND is performed between two binary vectors, the resulting vector

Table 2.6: Binary Representation of one SNP

Z	0	1	2	(...)	2	0
Z0	1	0	0	(...)	0	1
Z1	0	1	0	(...)	0	0
Z2	0	0	1	(...)	1	0

solely maintains ones in the entries where both the vectors on which the AND is performed are also ones. Naturally, all other entries will be filled with zeros. As such, since the entries of the contingency table are given by the number of ones that exist in the 3^K resulting vectors, each value can be obtained by performing a population count (PopCount) operation over the resulting vectors. Specifically, each entry of the contingency table of the pair-wise interaction between two SNPs, SNP A and SNP B, is given by counting the number of ones in each of the nine vectors, as a result of the bit-wise AND between all the combinations of the three vectors that define SNP A are the three vectors that define SNP B. Therefore, the entry n_{02} of the contingency table is obtained by performing a PopCount operation on the vector that results from the logic AND between the vectors A0 and B2.

2.2.3 Objective Functions

After the creation of the contingency tables (which represent the SNP-SNP interactions), there is a need to extract meaningful information from these tables. To do so, a statistical test needs to be performed using the generated data. The functions that are used to obtain relevant information from the generated contingency tables are often referred to as objective functions, and are used to evaluate the significance that each interaction has on the phenotype.

Given the variety of different objective functions used in epistasis detection and the lack of consensus among authors on the right one to use (due to the number of different ways that epistasis detection can be performed), there is no standard function or set of functions to use in epistasis detection. One of the most popular objective functions is the chi-square test (χ^2) [38], which is the test that commonly guides ant colony optimization algorithms [39]. There are also objective functions based on information theory [40, 41], such as mutual information [22] and information gain [42]. Objective functions based on ROC-curves [43] and regression modules [11] are also used in epistasis detection. Due to its easy implementation in hardware, and to its wide spread utilization [44], [45], mutual information is the objective function that is adopted in this thesis.

Multi-objective methods (as opposed to single-objective methods), which rely on the results of more than one objective function, have seen a rise in their popularity as they are effective in minimizing the number of false positives and negatives by combining the results of two or more objective functions. However, multi-objective methods introduces further computational complexity.

2.2.4 Mutual Information

Mutual information is an information theory concept used to quantify the mutual dependence between two random variables and it can be defined using the entropy of the two random variables, such that:

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \tag{2.2}$$

In fact, mutual information is often used as an objective function for epistasis detection [40, 46], as the genotype and the disease (phenotype) can be defined as the random variables X and Y, respectively. The entropy of the genotype and disease are defined in equations 2.3 and 2.4, respectively, where n_x represents entries of a contingency table and N_x the total number of patients, and the indexes 0 and 1 represent cases and controls, respectively. In equation 2.3, the summation is done for all entries of the contingency table. Note that for a balanced dataset, containing the same amount of cases and controls results in $H(disease) = 1$, which simplifies the calculation.

$$H(X) = -\sum[P(n_0 + n_1) * \log_2(P(n_0 + n_1))] \tag{2.3}$$

$$H(Y) = -[P(N_0) * \log_2(P(N_0)) + P(N_1) * \log_2(P(N_1))] \tag{2.4}$$

Using equations 2.2, 2.3 and 2.4 and attending to the fact that $P(n) = \frac{n}{N}$, where N is the total number of patients in the data set, the mutual information for each contingency table is defined by the equations 2.5 and 3.10.

$$I(X;Y) = \frac{N[H(X) - H(X,Y)]}{N} - H(Y) \tag{2.5}$$

$$N[H(X) - H(X,Y)] = \sum[n_0 * \log_2(n_0) + n_1 * \log_2(n_1) - (n_0 + n_1) * \log_2(n_0 + n_1)] \tag{2.6}$$

The higher is the mutual information value of an interaction, the more likely that interaction is to influence the phenotype (disease).

2.3 Software Implementations of Epistasis Detection

While several software-based implementations for detecting epistasis have been proposed over the years, as stated in Section 2.2, this thesis is focused on approaches based on exhaustive search, as it provides the most accurate results. Such approaches have mostly targeted second-order epistasis detection and are deployed in parallel computing platforms, such as muti-core CPUs and GPUs. iLOCi [12], and BOOST [11], which can be implemented using both a multi-core CPU or a GPU, are examples of such approaches, whereas the latter makes use of a binary encoding of the SNPs as the one described in Section 2.2.2 and used in the architectures proposed in this thesis.

As the computational burden grows exponentially with the epistasis detection order that is being

performed, there are only a few implementations targeting exhaustive third-order epistasis detection [16, 47, 48]. As an example, the GPU application to perform exhaustive third-order epistasis detection proposed in [47], makes use of four NVIDIA GTX Titan GPUs to produce the results in 10 minutes when using a dataset consisting of 1000 patients and 10 000 SNPs. A more recent implementation proposed in [16], makes use of the capabilities of the tensor cores existent in modern GPUs, namely in the ones based on the Turing architecture [49], by using the same binary encoding of the SNPs described in Section 2.2.2, providing faster results in both second and third-order exhaustive epistasis detection.

At the time of writing, no publication regarding the implementation of an exhaustive forth-order epistasis detection is found, due to the very high execution times that would be produced when adapting existent implementations, or due to the complexity involved in adapting an existent implementation to perform forth-order epistasis.

2.4 FPGAs in Epistasis Detection

Due to their ability to exploit parallelism at all levels, and the advances made in their technology in the past years, FPGAs have proven to be successful in accelerating the processing of data in bioinformatics applications [18, 20, 50], such as in sequence alignment [51–53], evolutionary biology [54] or secondary structure prediction of genetic data [55].

FPGAs allow for the implementation of custom made hardware, tailored for the application, being only limited by the amount of resources available in the reconfigurable fabric. Therefore, FPGAs provide more flexibility than a traditional CPU or a GPU, and are much cheaper to deploy than an ASIC. FPGAs excel in applications where it is possible to extract high levels of parallelism, such as the creation of contingency tables in epistasis detection, as multiple blocks for processing data can be instantiated, processing huge amounts of data in parallel.

In fact, accelerators capable of performing the parallel execution of several operations, such as GPUs and FPGAs, are being successfully used to accelerate epistasis detection. In this section the latest FPGA-based and heterogeneous (FPGA + GPU) accelerators for second and third order epistasis detection using exhaustive search methods are presented.

2.4.1 Second-Order Epistasis on FPGA

In 2014, an architecture was proposed to perform pair-wise epistasis detection on a RIVYERA S6-LX150 accelerator, containing 128 Xilinx Spartan-6 LX150 FPGAs [21]. Such implementation resulted on a speedup of more than 285$\times$ when compared with an implementation running on two Intel Xeon quad-core CPUs @ 2.4 GHz (4 minutes vs 19 hours) with both implementations running the iLOCi objective function [12]. This speedup was obtained for a WTCCC (Welcome Trust Case Control Consortium) dataset consisted of 500 000 SNPs and 5 000 patients. The energy consumption of the FPGA based architecture was roughly 1% of the CPU based application, despite the fact that the design uses more power than the CPU that it is compared against (780W vs 260W), as the execution time is much lower.

Although the energy used by this design is only a small fraction of the energy used by the CPU-based application that it is compared against, it is still a high value, as the power consumption of the used accelerator is very high since it uses multiple FPGAs.

To accelerate the contingency tables creation, the authors proposed an architecture based on a systolic chain topology (an array of tightly coupled processing elements), where each Processing Element (PE) stores one SNP and processes it against the SNPs that are being streamed throughout all the PEs. At first, since all PEs need to be initialized, when the SNP data starts to be streamed throughout the PEs, if empty, each PE will store the SNP data in their Block Random Access Memory (BRAM). When PE is already initialized, it will process the SNP against the stored one and send the data to the next PE where the same process is repeated. A scheme of the implemented systolic chain, taken directly from [21], is represented in Figure 2.1. The rate by which the SNP data is streamed to the systolic array of processing elements is not specified in [21]. However, in a subsequent incremental solution proposed by the same authors [22], is possible to infer that only 2 SNPs are streamed on each clock cycle, as each SNP is coded in a two-bit representation, only 4 bits are streamed to the systolic array per clock cycles.

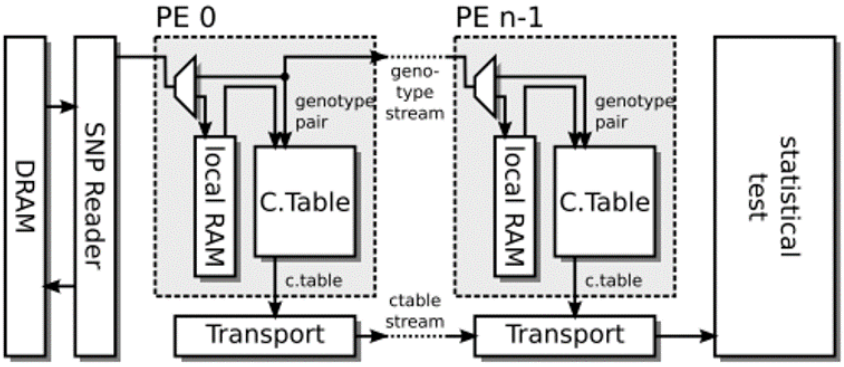

Figure 2.1: Systolic chain scheme for the creation of contingency tables, proposed in [21]

Figure 2.2 shows an example of the implementation used in [21] for a dataset with 6 SNP being processed by a systolic chain of 3 PEs. In the example, the black squares indicate the pairs of SNPs yet to be processed, the gray squares indicate the pairs that are being processed, and the white squares the pairs that were already processed.

In step 1, all the PEs are uninitialized and the SNP data starts to be streamed. In step 2, PE 1 stores SNP0, and does not send the data to the next PE (as it was empty). When in step 3, the data referring to SNP 1 is streamed to the systolic chain, PE 1 processes it against SNP 0 (stored in its BRAM). At this point, as the PE 0 is already initialized, it sends SNP 1 to the next PE, which will in turn store it in its BRAM. At the end of the fourth step, all PEs are initialized, storing the data of SNP 0, 1, and 2. In steps 5 and 6, SNP 4 and 5 are processed against the SNPs stored in each PE, creating as many contingency tables as there are PEs (three in this case). In step 7, all SNP have been processed against the ones stored in the PEs, as such, the SNP data stored in the PEs is discarded. In steps 8 and 9, the PEs are initialized again, but instead of streaming the data referring to SNP 0, the process starts with SNP 3, as it was the first not to be previously stored in any PE.

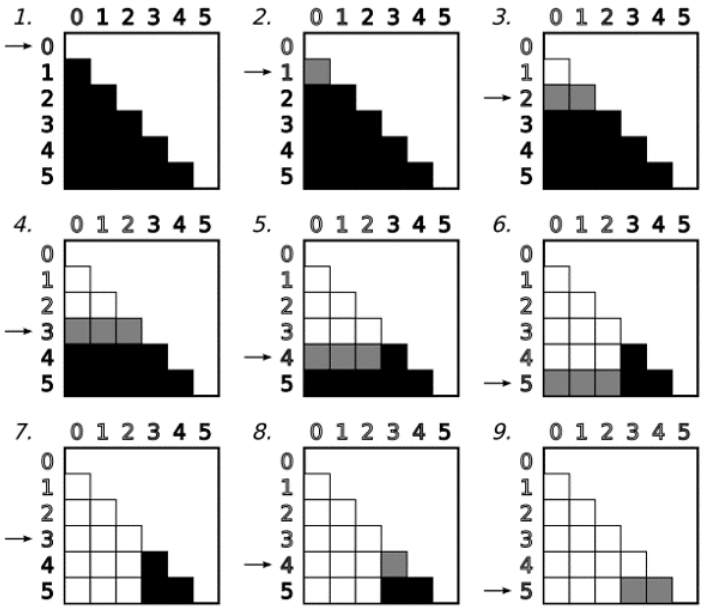

Figure 2.2: Processing of a 2nd order interaction of 6 SNPs in 3 PEs [21]

The contingency tables are preprocessed in each PE in such way that only the values that are relevant to the calculation of the objective function (a, b and c from the equation 2.7, where n_{xy} are entries of a contingency table) are sent to a statistical unit where the iLOCi statistical test is performed according to the equation 2.8.

$$\begin{cases} a = n_{00} - n_{02} + n_{22} \\ b = \sum_j n_{0j} + \sum_j n_{2j} \\ c = \sum_i n_{i0} + \sum_i n_{i2} \end{cases} \tag{2.7}$$

$$p = \frac{a}{\sqrt{bc}} \tag{2.8}$$

The resulting value from equation 2.8 is calculated both for cases and controls, and their difference is then calculated according to equation 2.9, which corresponds to the final value of the iLOCi for each pair of SNPs. The higher the p^{diff}, the greater the probability of interaction is, the n highest values of p^{diff} are stored in a sorted list.

$$p^{diff} = \left| p^{case} - p^{control} \right| \tag{2.9}$$

2.4.2 Third-Order Epistasis on FPGA

An implementation to detect third-order epistasis in an FPGA using mutual information as objective function was proposed in [22]. The design was implemented in two different FPGAs for comparison, a Virtex7-VX690T and a lower end Kintex7-K325T resulting in a speedup of $363\times$ and $181\times$, respectively, when compared with an application running in an Intel Core-i7 Sandy Bridge @3.20 GHz 6-core CPU,

for a dataset with 10 000 SNPs and 5 000 samples. The relation of 2x between the speedups obtained for the two FPGAs was expected, as the Virtex-7 device contains almost twice the amount of resources as the Kintex-7 device.

The implementation is very similar to the one detailed in [21], as the creation of the contingency tables is done essentially in the same way. However, as the order of interactions increases from 2 to 3, the complexity of each PE and the size of each table also increases. In this solution, each PE now stores data referring to two SNPs (to process against the SNP being streamed) to create the contingency tables for third-order interactions. As the data of two SNPs is stored in each PE, this architecture requires twice the amount of BRAMs in each PE than the second-order solution proposed in [21].

Additionally, when compared to [21], the FPGA, adopted in this solution allows for an increased width for the data bus that streams the SNP data from memory to the PEs, resulting on the simultaneous streaming of eight SNPs per clock cycle, encoded in a two bit representation, meaning that 16 bits are sent every clock cycle. This improvement in speed combined with bigger contingency tables introduces strain on the transport system and in the statistical unit where the objective function is calculated. As such, to prevent the introduction of stalls, a chain of PEs can only have 17 PEs, that number does not exploit all the available resources of the FPGAs. To use the maximum number of available resources, several chains of PEs were implemented, each one with its statistical test unit. In the Virtex 7 FPGA, the available logic allowed for the implementation of 204 PEs distributed in over 12 chains, the Kintex 7 FPGA managed to fit 102 PEs divided into 6 chains. Figure 2.3 depicts the organization of the implemented systolic chain.

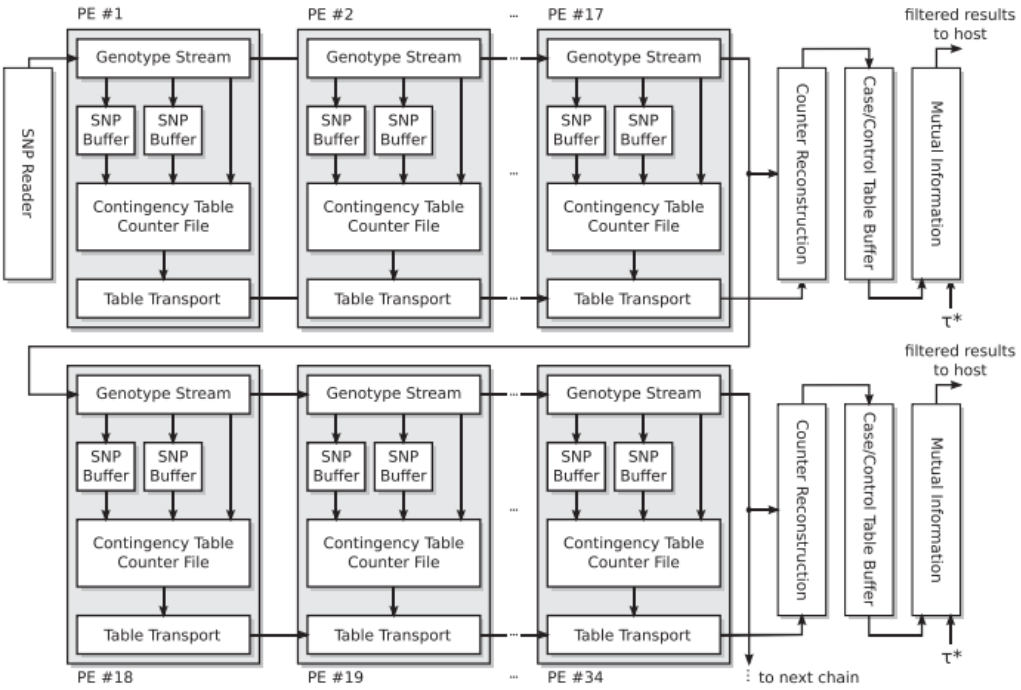

Figure 2.3: Systolic chain scheme for $K = 3$ epistasis detection proposed in [22]

Despite each contingency table for a third-order interaction having 27 entries for cases and 27 for controls, only 20 of those need to be calculated in the PEs, as the other 7 can be inferred from the

calculated values, provided that the total number of cases and controls is known. As such, each PE is only required to calculate 20 entries of the contingency tables both for cases and controls, saving resources that are used to instantiate more PEs. A counter reconstruction unit is included at the end of each chain to calculate the omitted values.

In this implementation [22], the objective function used is the mutual information, the calculation function (given by $N(H - H')$) is done in a pipelined fashion for every 27 entries of the contingency tables. Figure 2.4 shows a scheme of the implementation of the mutual information calculation and comparison with a threshold, where the inputs are entries of the contingency tables. Since the architecture is pipelined, one input can be provided at every clock cycle, hence, taking 27 clock cycles plus latency of the unit that performs the calculation to calculate the mutual information value for each contingency table (i.e., for each triplet of SNPs).

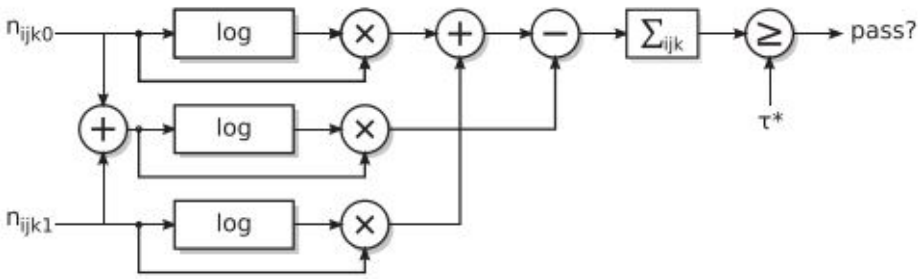

Figure 2.4: Block diagram for the calculation of $N(H - H')$ proposed in [22]

Although both architectures from [21] (see Section 2.4.1) and [22] represent important breakthroughs in FPGA-based epistasis detection implementations, they are also limited by lacking the ability to run any dataset size, as both architectures are specifically dimensioned for a particular number of patients. When using a dataset with fewer patients, the contingency tables will be generated in each PE faster, which will cause even more strain on the table transport system, as such, a new design would have to be devised and implemented for a different number of patients. Such a drawback makes both solutions quite limited in terms of applicability. Hence, there is an opportunity to explore new and more sophisticated solutions that deploy more efficient and versatile FPGA-based epistasis detection accelerators.

2.4.3 Heterogeneous implementations (FPGA +GPU)

In an effort to improve the performance of FPGA-based solutions, heterogeneous implementations using a host computer and an FPGA-GPU system have also been proposed for both second and third-order epistasis detection [25, 56, 57]. All three implementations are similar when it comes to the way the processing is divided, the contingency tables are created in the FPGA, in the same way that is done in [21] for second-order and [47] for third-order, (detailed in Sections 2.4.1 and 2.4.2, respectively). The created contingency tables are streamed to the GPU, where the objective function for each of the tables is calculated. This allows for the implementation of more PEs in the FPGA, as the resources that would be used to implement the objective function are essentially free. The combinations are filtered in the

GPU according to their objective function result and the result is sent to the Host. A conceptual scheme of the system is illustrated in Figure 2.5.

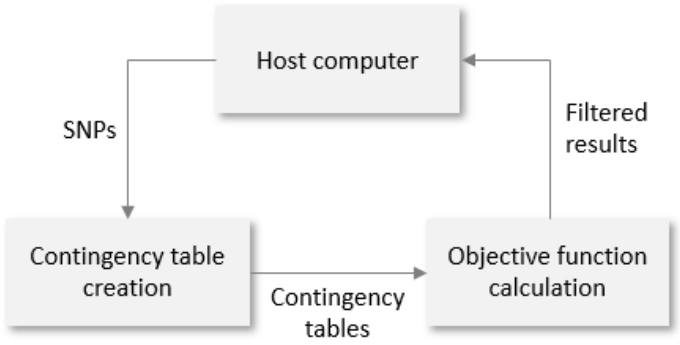

Figure 2.5: Heterogeneous architecture using an FPGA and a GPU

The main advantage of this implementation is the computation of more complex objective functions that may not be suited for an efficient FPGA implementation. By doing so, there are more available resources in the FPGA to implement more PEs, so that more contingency tables can be created in parallel. However, since there is no direct connection between the FPGA and the GPU, the data transfer corresponding to all contingency tables, needs to be done through the host computer. This introduces a significant overhead that gets worse the higher the epistasis detection order is, as more and bigger contingency tables are produced, and for fewer patients, as each table is produced faster. This solution also introduces more complexity to the configuration/programming of the devices when compared to approaches solely based either on FPGAs or GPUs, as they are done using different languages and different software, which creates an additional barrier to scale the implementation to detect higher order epistasis.

Second-Order Implementation

An heterogeneous implementation using high-end hardware to detect second order epistasis is proposed in [25]. this system was implemented using a Xilinx Kintex UltraScale KU115 FPGA and an an Nvidia Tesla P100 GPU, both connected to the host computer through two distinct PCI Express Gen3 interfaces.

The creation of the contingency tables is done in the FPGA in the same manner as described in Section 2.4.1. 408 PEs divided in two chains are implemented. The contingency tables are sent to the GPU though the host computer where the objective function (logic regression) is calculated in the GPU for each contingency table. The results are then filtered according to their score and sent to the host computer. The performance of the system starts to drop for less than 10000 patients, as the contingency tables are created faster and, therefore, more bandwidth is necessary to send the tables to the GPU.

Third-Order Implementation

A low-cost heterogeneous implementation to detect third-order epistasis is detailed in [57]. I makes use of a Virtex-7 690T FPGA and an Nvidia GeForce 780 Ti GPU in a host computer, equipped with an Intel Core i7-4790K quad-core processor at 4GHz, coupled with 32GB of DDR3 RAM. The creation of the contingency tables in the FPGA is done in the same way as in [47], with the some improvements made to the pipeline. The created contingency tables are sent to the GPU where the omitted entries of the contingency tables are inferred before an information gain objective function is calculated (contrasting to the simpler mutual information function used on the FPGA only implementation). The results are then sent to the host where they are filtered and sorted in a min-max heap that stores a user defined number of best results.

The speedups obtained with this implementation are between $70\times$ and $90\times$ for datasets with 5000 and 40 000 samples, respectively when compared with a host only application, and roughly $4.9\times$ when compared with a GPU only implementation.

2.5 Discussion

The massive amount of contingency tables created to perform high-order epistasis detection in a large dataset using exhaustive search methods, and the fact that all those tables are created using the same operations over different SNPs, means that each contingency table is created independently. Therefore the creation of contingency tables greatly benefits from the exploration of the existing data parallelism. By using an FPGA, it is possible to take advantage of such parallelism, as the implemented circuit is tailored to the task. Due to recent advances in FPGA technology, the number of resources available in modern FPGAs allows for designing a circuit that is able to exploit the existent parallelism in the creation of contingency tables. This means that several contingency tables can be created in parallel at the same time (depending on the size of the FPGA, which is not possible (to the same extent) when using a single traditional CPU. As demonstrated by the state-of-the-art implementations, recent FPGAs allow for the creation of hundreds of contingency tables in parallel. As the FPGA technology keeps evolving, it is safe to assume that in the future even more combinations can be processed in parallel at even higher clock frequencies, creating an opportunity to preform higher-order exhaustive search epistasis detection in FPGAs.

Despite providing promising results, the existent FPGA implementations (detailed in Section 2.4) have limitations that impact their applicability for higher-order epistasis detection. The second-order implementation (described in Section 2.4.1) is implemented in an accelerator containing 128 FPGAs, which has a very high power consumption (780W). The third-order architecture (described in Section 2.4.2) stores two SNP in each PE to process against the SNP that is streamed through the systolic array to compute third-order interactions. Hence, if this architecture were to be changed to perform higher-order epistasis detection, the utilization of BRAMs in the FPGA would a limiting factor to the number of PEs that can be implemented, as the utilization of BRAMs per PE increase with the order

of interactions, limiting the scalability of this solution for higher-order epistasis detection. The state-of-the-art architectures for both second and third-order are designed for a specific range of patients, which further limits their applicability.

The architectures proposed in this thesis aim to address the main limitations of the state-of-the-art implementations by providing an easy way to deploy FPGA-based accelerators for any order of interactions. The proposed accelerator achieves similar execution time as the state-of-the-art implementation using a significantly less amount of power, as only one FPGA is used to implement the design. The method proposed in this thesis solves the problem of BRAM utilization when adapting the existing architectures to higher-order epistasis by changing the order by which the combinations are processed, in a way that only in the first PE stores more than one SNP. Therefore, the utilization of BRAMs per PE is the same regardless of the order of interactions (except in the first PE). Besides addressing the limitation of the existing FPGA implementations, the proposed architectures also provides higher performance and energy efficiency.

The architecture proposed in this thesis relies on a single-objective method for epistasis detection, using the mutual information as the only objective function, as it can be efficiently implemented in an FPGA, as it can be calculated using simple operations. The implementation of a multi-objective system, which uses more than one objective function would have a meaningful impact in the performance of the proposed architectures, as the implementation of an objective function utilizes a considerable amount of the FPGA resources available.

2.6 Summary

This Chapter, started by providing a background on fundamental genetics concepts, with particular emphasis on the organization of the DNA, essential to understanding the mutations that can occur in the human genome, particularly the SNPs. These concepts were followed by an introduction to GWAS and epistasis, and their importance in the identification of the DNA mutations that can lead to certain diseases.

After this introductory stage, the main methods used to detect epistasis were detailed, with particular emphasis on exhaustive search methods. The concept of contingency tables and objective functions, used throughout this thesis, were also introduced, with special focus on the creation of contingency tables when using of a binary notation to represent the SNPs, and on the use of mutual information as an objective function.

Next, the state of the art in implementations of exhaustive search epistasis detection in software was presented, followed by detailed descriptions of the most recent FPGA-based approaches, as well as heterogeneous systems combining FPGAs and GPUs in host-based systems.

The chapter was concluded by a discussion on the main limitations of current epistasis detection accelerators (particularly, those implemented in FPGAs) and by detailing the main opportunities explored in this Thesis.

Chapter 3

Specialized Accelerator Architecture for High-order Epistasis Detection

In this thesis, a new epistasis detection accelerator architecture for FPGA platforms is proposed. The proposed accelerator is generated by an especially devised method that allows the parameterization of the architecture for any order of interaction and any number of patients and SNPs. Such a solution presents a two-fold benefit when compared to existing FPGA-based solutions, by not only enabling the deployment of specialized accelerators for higher than third-order epistasis detection in FPGAs, but also by allowing a design-time parameterization to support an arbitrary number of patients, according to the targeted dataset.

The proposed general accelerator architecture and the system organization are thoroughly described in the following sections, by first providing an overview of the whole system, followed by specific design decisions, architecture details, and implementation requirements for each of its components. This Chapter is concluded with the proposal and description of dedicated accelerator architecture, derived from the general parameterizable architecture, specially optimized to the specific requirements of second-order epistasis detection.

3.1 System Overview

This section provides an overview of the proposed system for the acceleration of epistasis detection. Section 3.1.1 outlines the general organization of the hardware accelerator and its interaction with the software application by presenting their main components. The data representation and details regarding the data transfer management between the FPGA-based accelerator and the host General-Purpose Processor (GPP) are provided in Section 3.1.2.

3.1.1 Architecture Overview

The proposed system is composed of an FPGA-based accelerator, parameterizable for any number of patients and scalable with the order of interactions, to create contingency tablesfor the SNP combinations under evaluation and calculate their mutual information scores. The accelerator is connected to a host GPP running a dedicated software application, which is responsible for the controlling of the processing order and for managing the data transfers to/from the FPGA's on-board memory. The accelerator can either be connected to the GPP in a tightly coupled system (e.g., inside the same SoC), or by connecting a board containing an FPGA to a host computer (e.g., via a PCI-Express bus). This thesis is focused on a tightly coupled system, where the FPGA and the GPP are included in the same SoC. Nonetheless, the proposed architecture is abstracted from the existent interface between the host and the FPGA accelerator, so that it can be implemented in different systems.

Figure 3.1 shows an overview of the proposed system. The SNP data, sent by the Host through the high-performance interconnection, is fed to a systolic array of functional units that generate partial contingency tables (as only the G0 and G1 vectors of each SNP are streamed). Next, the partial tables are completed in the reconstruction unit before their mutual information value is calculated. The SNP combinations that produce higher mutual information values are stored. As depicted in Figure 3.1, the reconstruction units and the following blocks can be shared among several Contingency Table Units (CTUs). Ideally, they could be shared between all the CTUs in the systolic array, however, the level of resource sharing is (as detailed in Section 3.2.4) dependent on how fast the partial contingency tables are created. The *count last* block, at the beginning of the systolic array, receives the data that is streamed from the host and propagates it to the systolic array along with some control signals that it generates. The unit is therefore responsible for the control of the systolic array of CTUs.

Accordingly, the architecture can be divided into two main stages. The partial creation of the contingency tables in the systolic array (and their subsequent reconstruction), and the mutual information calculation of the generated tables.

3.1.2 SNP Data Representation and Streaming

The proposed accelerator accepts datasets stored in the binary representation detailed in Section 2.2.2 by streaming SNP at a time following a specially devised representation (detailed below).

The accelerator's memory accesses are controlled by the software application running in the host GPP, by defining the size of the data transfer (number of words), the first SNP to send (base memory address), and reset and re-initialization signals (detailed in Section 3.2.1). The width of the interface between the memory where the dataset is stored and the FPGA (referred to as R), can also be configured prior to the design synthesis. From those R bits, the most significant half is used for the cases while the least significant half is used for the controls. This approach is valid for a balanced dataset (that the number of patients classified as cases is the same as the number of control patients). For an unbalanced dataset, the ratio of bits that is used for cases and controls in each word needs to be adjusted depending on the ratio of cases and controls in the dataset.

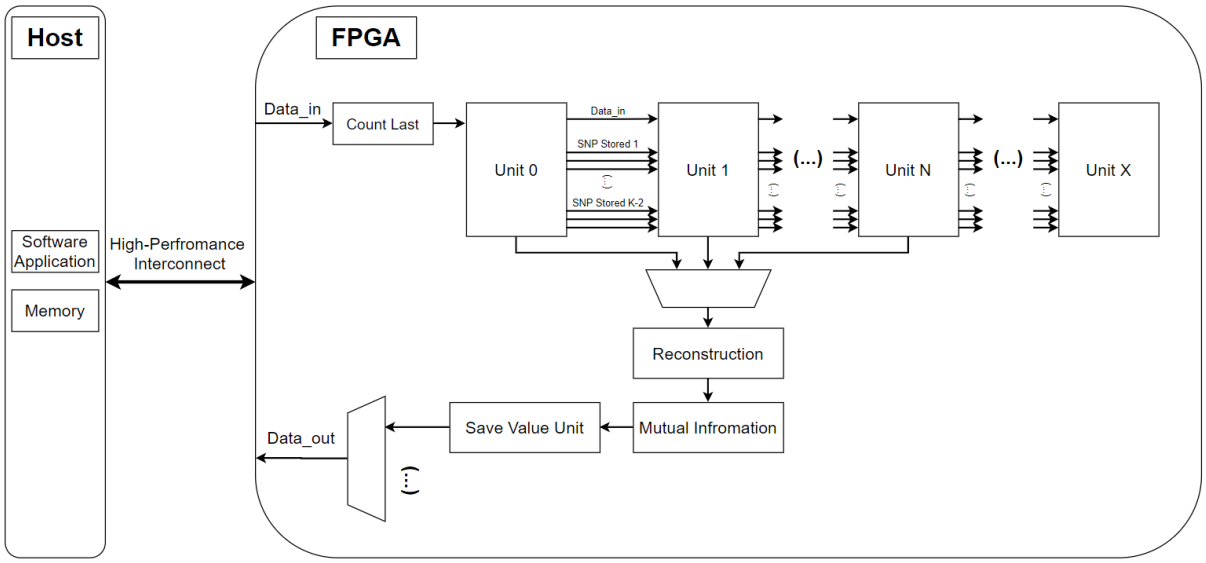

Figure 3.1: System and generic architecture overview

Additionally, to stream the SNPs data to the FPGA, the number of words that are streamed for each SNP must be even, as the first word corresponds to the genotype zero vector (G0) of the first $R/2$ cases and $R/2$ controls, and the second word is used to send the genotype one vector (G1) of the same patients. The genotype two vector (G2) is not stored in the memory nor streamed to the FPGA since it can be generated when needed by preforming a bitwise NOR between G0 and G1 Hence, the number of words used to stream one SNP is given by Equation 3.1 where $\#Patients$ is the number of patients in the dataset and R is the width of the interface.

$$\#Words\_SNP = \left\lceil \frac{\#Patients}{R/2} \right\rceil + \left(\left(\frac{\#Patients}{R/2} \right) \mod 2 \right) \tag{3.1}$$

Whenever the number of patients is not divisible by $R/2$, dummy patients are added to that dataset (automatically ignored by the architecture), until a number of patients divisible by $R/2$ is produced. This is contemplated in Equation 3.1 by the ceiling of the first term. Also, if the first term of the equation does not produce an even number of words, another word is added, i.e. $R/2$ dummy patients are sent. Since dummy patients are always added to the last pair of words, there might be an excess of data being streamed to the FPGA in the last two words of each SNP. As such, those positions are filled with zeros in both G0 and G1. However, by doing so, G2 that is generated in the FPGA will have a logic one in those excess positions that must be ignored by the units that create the contingency tables so that the excess number of bits that are streamed are not interpreted as valid patients.

The data stream goes directly to the *count last* unit, as depicted in Figure 3.1. As mentioned before, this unit serves as a controller for the systolic chain. This control is attained by counting the number of words that are received and comparing the count value against the number of words per SNP, signaling the last word of each SNP. Both the value of the counter and the flag that signals the last word from each SNP are propagated throughout the systolic chain at the same time as the word itself. Although the *count last* unit introduces a latency of 1 clock cycle, only one counter is needed to coordinate the

entire systolic array, and the added latency is hidden by the pipelined architecture of the accelerator.

As described in Section 3.3.3, each *save value* unit, stores the X best mutual information values, represented in the single-precision floating-point format, and the combinations corresponding to those mutual information values by storing the K SNP identifiers that compose the combination, each of them in a 32-bit word. Consequently, after the processing of all the possible combinations, each unit streams to the host X mutual information values and $(X * K)$ 32-bit SNP identifiers. Therefore, the number of words that are streamed to the host is given by Equation 3.2, where R is the width of the interface in use, and N_save_units the number of save units implemented in the design. The combinations are then sorted in the host GPP by their mutual information value and the X best combinations are presented to the user.

$$\#Words\_res = \left\lceil \frac{32 * (1 + K)}{R} \right\rceil * X * N\_save\_units \tag{3.2}$$

3.2 Systolic Accelerator for Contingency Table Creation

The proposed systolic array to accelerate the calculation of the contingency tables works in a similar fashion as the one proposed in [21] (previously described in Section 2.4.1). However, it can be easily adapted to work for any order of interactions, by changing each CTU and the bandwidth between them.

To create contingency tables for every possible combination, the dataset is streamed to a systolic chain of units several times. Each time the necessary data is streamed to the FPGA is referred to as a round. In each round a different number of SNPs are streamed to the FPGA to ensure that all possible combinations are tested and no combination is repeated. The number of SNPs that are streamed in each round is controlled by the software application running in the host GPP.

3.2.1 Data Reorganization and Processing Order

As explained in Section 3.1.2, the interface that is used to stream the dataset to the FPGA-based accelerator is controlled by the software application running on the host. The software is responsible for choosing the first SNP to be streamed in that round (by setting the base memory address), the number of SNPs that are to be streamed in that round (by choosing the size of the data transfer), and the management of the re-initialization of the systolic array (by defining the re-initialization mode), according to the process described bellow.

Between each round, the units of the array are re-initialized, changing the SNP that is stored in each one, so that each unit always produces a different combination in each round. For the first unit, there are $(K - 1)$ different possible re-initialization modes (re_init_1, ..., $re\_init\_(k - 1)$) where the number of the mode equals the number of BRAM pairs (with are used to store the SNPs in each CTU) that will be updated at the beginning of the next round for the cases and the controls. In the following units, both pairs of BRAMs, one for the cases and the other for the controls, are always updated at the beginning of a round whichever re-initiation mode is chosen.

Re-initialization	Unit 1	Unit 2	Unit 3	Re-Initialization	Unit 1	Unit 2	Unit 3
	0-				06	16	26
	01	1-		Re_init_0	3-		
	02	12	2-		34	4-	
	03	13	23		35	45	5-
	04	14	24		36	46	56
	05	15	25				

Table 3.1: Example for K=2 with 3 units and 7 SNPs

The lower re-initialization modes are always prioritized, which means that $re\_init(N)$ is only done when all the possibilities resulting from using $re\_init(N-1)$ have already been tested. This scenario occurs when the number of SNPs that can be stored in the CTUs is higher than the number of SNPs streamed in that round, this number corresponds to the sum of the number of BRAMs to be updated in that round in the first unit (which depends on the last re-initialization mode that was used), with the number of CTUs minus 1, as all the units, except for the first, only store one SNP (which change in every round).

Table 3.1 shows an example for second-order epistasis detection, using three units and a dataset containing seven SNPs. The combinations that are produced by each unit and the re-initialization mode that is used at the end of each round. As shown in the example, only one SNP is stored in each unit, which is the first of the produced combination. The second SNP is streamed directly from the communication interface and is sent directly to the next unit without being stored.

As this is an example of an exhaustive search for second-order, only one re-initialization mode is used, which resets the only SNP that is stored in each unit. In the first round, every SNP is streamed from the host to the systolic array. Since there are three available units, SNPs 0, 1 and 2 are stored in units 1, 2 and 3, respectively. As all the SNPs are streamed, all the combinations that can be produced using the SNPs that are stored in the units are tested. As such, in the second round, only the SNPs that have not been stored in any unit are streamed. The second round is also the last round to be made, as the last combination, $(5, 6)$, is tested in that round.

Table 3.2 also shows the combinations that are produced by each unit, and the re-initialization mode that is used at the end of each round, for a third-order exhaustive search using three units and a dataset containing seven SNPs. Accordingly, two re-initialization modes are used, re_init_0 which resets the first SNP stored in the first unit, and re_init_1 which resets both SNPs stored in the first unit. All the re-initialization modes reset the only SNP stored in each of the remaining units.

The first SNP of the generated combination is the one that is only stored in the first unit, the second SNP is the one that is stored in the unit producing the combination, and the third SNP is streamed from the host memory and sent directly to the next unit.

In the first round, every SNP is streamed to the systolic array. Since, at the end of the round, there are still combinations to be tested containing the SNP 0, the first re-initialization mode is used. Accordingly, SNP 0 is kept in the first unit and the dataset is streamed from the on-board memory starting on SNP 4, as it was the first SNP not to be stored in any unit. When the second round finishes, all possible

Re-initialization	Unit 1	Unit 2	Unit 3	Re-Initialization	Unit 1	Unit 2	Unit 3
	0–				126	136	146
	01–			re_init_1	15-		
	012	02-			156	16-	
	013	023	03-	re_init_2	2–		
	014	024	034		23-		
	015	025	035		234	24-	
	016	026	036		235	245	25-
re_init_1	04-				236	246	256
	045	05-		re_init_2	3–		
	046	056	06-		34-		
re_init_2	1–				345	35-	
	12-				346	356	36-
	123	13-		re_init_2	4–		
	124	134	14-		45-		
	125	135	145		456	46-	

Table 3.2: Example for K=3 with 3 units and 7 SNPs

combinations containing SNP 0 have been tested, therefore, the second re-initialization mode is used. The dataset is then streamed from SNP 1, which is stored in the first unit. All combinations containing SNP 1 are tested before the second re-initialization mode is used again (before the fifth round). The process ends when the combination 456 is tested in the first unit, in the seventh round.

The described process can be easily adapted for any order of interactions, by using more re-initialization modes, and for any number of CTUs and SNPs. The number of patients of the dataset is not relevant for this process, as it only impacts the throughput of the CTUs, the number of implemented reconstruction units, and the width of the registers and buses that store and transfer the contingency tables, respectively.

For second-order interactions, the number of SNPs that are streamed in each round is always decreasing and is given by Equation 3.3.

$$\#SNPs\_in\_round = \#SNPs - (Round \times \#Units) \tag{3.3}$$

The SNPs that have already been stored in a CTU do not need to be streamed again. The last round occurs when the number of SNPs to be streamed is less than the number of implemented CTUs.

3.2.2 Contingency Table Units Architecture

The units that create the partial contingency tables for each combination are, similarly to [22], arranged in a systolic array. However, contrarily to what is done in the third-order implementation from [22], where each unit stores the first two SNPs that are received and sends the rest of them to the next unit, in the proposed design, each unit only stores one SNP, except for the first unit that stores $(K-1)$ SNPs. This alteration prevents $(K-2)$ SNPs to be stored in all the units except in the first which reduces the utilization of BRAMs that could limit the number of CTUs that can be implemented for $K > 2$.

Figures 3.3 and 3.2 illustrate half of the datapath of CTU 1 and CTUs [2,..., $\infty - 1$], respectively, for any order of interactions. The datapath shown in the figures only illustrates half of the datapath, corresponding to the creation of contingency tables for the cases, as the other half, that is responsible to create the contingency tables for the control, works exactly in the same manner. The difference between the first CTU and the rest is the number of SNPs that they store and send to the next unit, and that difference only affects the first of the five pipeline stages that compose the units as well as the number of input ports that each unit have. Accordingly the CTUs (depicted in Figure 3.2), only one SNP is stored (exept in the first), since the $(K-2)$ SNPs that are stored in the first unit are being streamed throughout the systolic array at the same rate as the SNP that comes through the *data_in port*. As such, every unit has access to different combinations of SNPs of the K^{th} order, having to store only one SNP, requiring each unit to have $[1 + (K-2) * 3]$ inputs, and output ports, and additional output ports to sent the contingency tables to the reconstruction unit.

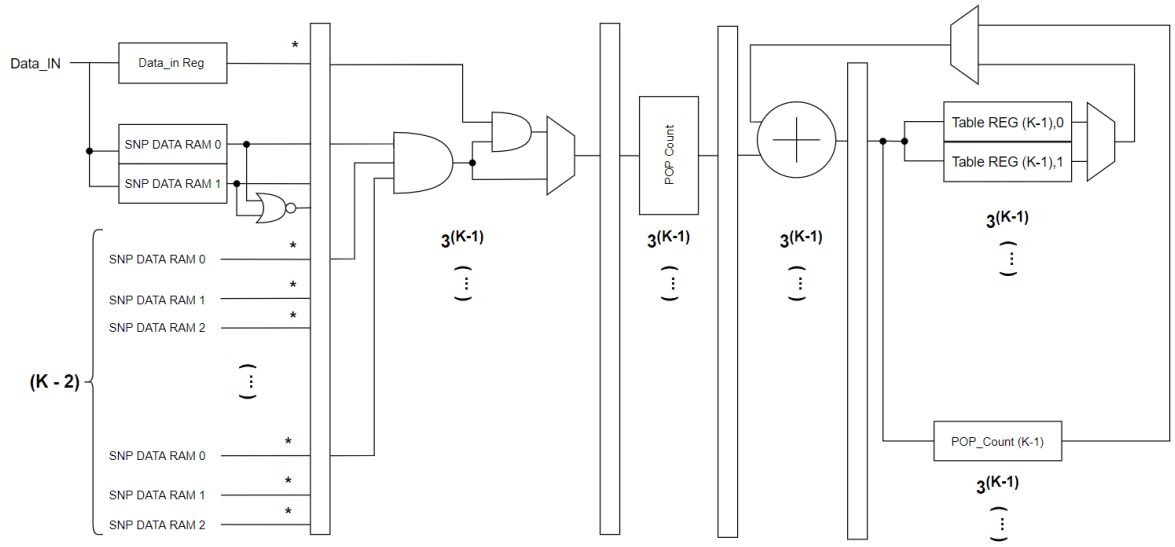

Figure 3.2: Datapath for half of the CTUs

All the CTUs have a latency of five clock cycles between the arrival of a word and the calculation of the partial values of the contingency table corresponding to that word. The number of cycles needed for each unit to produce a complete table is equal to the number of words that compose one SNP, as the units accept one word per clock cycle, plus the five cycles needed to calculate the last partial result. As the unit works as a pipeline, the latency of the unit is only relevant for the creation of the first partial contingency, the throughput of the unit is one table in the same number of clock cycles as the number of words that are needed to stream one complete SNP. Details regarding the implementation of the PopCount operator and the adders that are present in the CTUs are presented in Section 4.3.1.

The functioning of the CTUs is divided into three phases: (i) the creation of a complete contingency table for $(K-1)^{th}$-order (necessary to the reconstruction of the missing contingency table entries); (ii) the generation of the third of the contingency table that depends on G0 of the SNP that is streamed from the memory; (iii) the generation of the third of the contingency table that depends on G1 of the same SNP. The second and third stages are executed alternately.

First Phase

The first phase of the execution of a CTU is the creation of a complete contingency table for $(K-1)^{th}$-order, whose entries are referred to as "$n_{(K-1)}$". This process is initiated while the $(K-1)^{th}$ SNP is being streamed from memory and stored in the first stage of the unit's pipeline. As the complete contingency table is to be generated, a NOR port is used to generate G2 of the stored SNPs.

In the second and third pipeline stages, all the $3^{(K-1)}$ possible combinations between the (K-1) stored SNPs are processed (by performing an AND between G0, G1, and G2 of the stored SNPs) and the subsequent PopCount. The results of this stage correspond to partial entries of the $(K-1)^{th}$-order contingency table. As such, in the fourth and fifth stages, the values are accumulated until all the patients that compose the SNP have been streamed from the memory, while the accumulated values are stored in the "POP_count (K-1)" registers. Therefore, at the end of the first phase, these $3^{(K-1)}$ registers will store an entire $(K-1)^{th}$-order contingency table.

Figure 3.3 represents the datapath of CTU emphasizing the datapath followed in the fist execution phase.

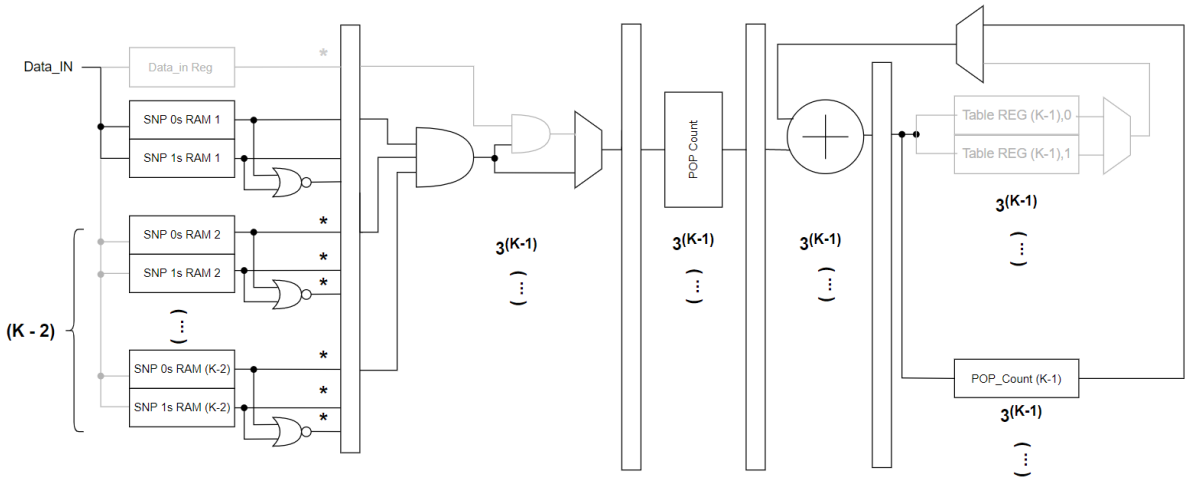

Figure 3.3: Datapath for half of the first CTU emphasizing the first execution phase

Second Phase

The second phase of execution of a CTU is initiated when the last SNP of the combination starts to be received through the "Data_IN" port. In the second pipeline stage, the first set of the G0 vector of the SNP is processed against the first set of the G0, G1 and G2 vectors of the stored SNPs. This generates $3^{(K-1)}$ vectors that are PopCounted in the third stage. As the results of the PopCount correspond to partial entries, the values are stored in the "Tab REG (K-1), 0" set of registers (in the fifth stage), and accumulated with the next set of values corresponding to the G0 of the streamed SNP. However, as the second word that the unit receives refers to the first set of the G1 vector of the streamed SNP, the accumulation is only done every two clock cycles.

At the end of the execution phase, the "Tab REG (K-1), 0" set of registers hold the third of a contingency table that is dependent on the G0 vector of the last SNP of the combination "$n_{(K-1),0}$".

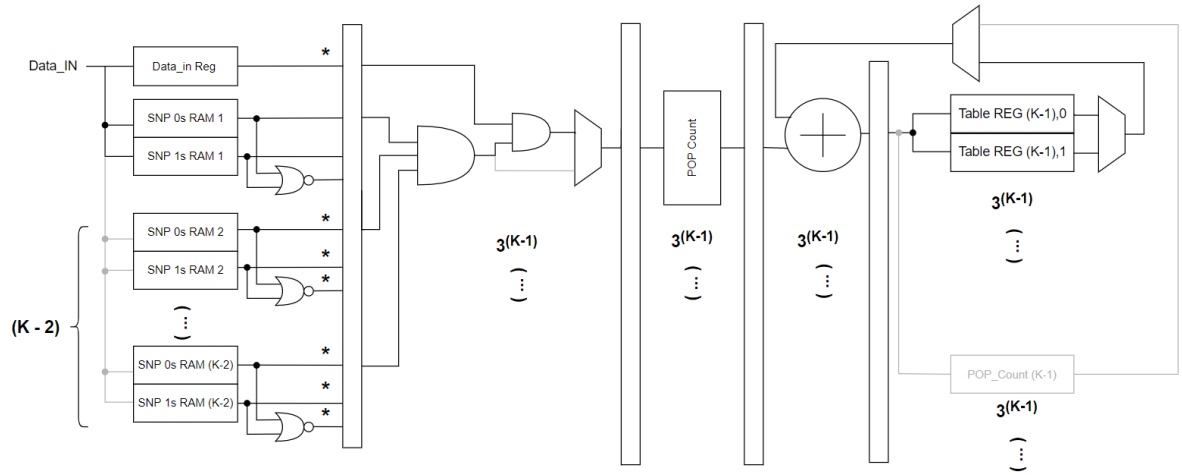

Figure 3.4: Datapath for half of the first CTU emphasizing the second and third execution phases

Third Phase

As partial G0 and G1 vectors of a SNP are streamed alternately, the second and third phases are also executed alternately. The third execution stage of a CTU is similar to the second. However, the values produced in this stage are the contingency table entries dependent on the G1 vector of the last SNP of the combination "$n_{(K-1),1}$". The partial values produced in this stage are stored in the "Tab REG (K-1), 1" set of registers.

Figure 3.3 represents the datapath of CTU emphasizing the datapath followed in the second and third execution phases. To alternate between both execution phases one of the registers in the fifth pipeline stage is deactivated and the following multiplexer is controlled accordingly.

3.2.3 Reconstruction Units Architecture

As already mentioned, the values produced in the CTUs, do not correspond to the complete contingency table of the combination. As such, the missing values are calculated in the reconstruction units. Figures 3.5 illustrates the datapath of half (cases or controls) of the reconstruction units generated using the general method for any order of interactions (any value of K).

Since G2 from the SNPs that are streamed from the communication interface is never generated, the entries of the contingency table that depend on that vector (represented by $n_{[(K-1),2]}$), are not generated in the CTUs. To calculate the missing values, the contingency table of the $(K-1)^{th}$ combination (generated in the CTUs first phase of execution), is used in accordance with the Equations 3.5 and 3.4

$$n_{(K-1)} = n_{[(K-1),0]} + n_{[(K-1),1]} + n_{[(K-1),2]}, \tag{3.4}$$

$$n_{[(K-1),2]} = n_{(K-1)} - (n_{[(K-1),0]} + n_{[(K-1),1]}), \tag{3.5}$$

where $n_{(K-1)}$ represents the entries of the contingency table for the $(K-1)^{th}$-order combination, and $n_{[(K-1),0]}$, $n_{[(K-1),1]}$ and $n_{[(K-1),2]}$ are entries of the contingency table for the K^{th} order combination,

resulting in an AND between $n_{(K-1)}$ and vectors G0, G1 and G2 of the next SNP, respectively. As such the equality expressed in Equation 3.4 is always valid.

Equations 3.4 and 3.5 can be more easily understood by taking in account an example for a third-order contingency table. An entry of a second-order contingency table, such as ”10”, when combined with another SNP results in three entries of a third-order contingency table, ”100”, ”101” and ”102”, depending on the vectors G0, G1 and G2 of that SNP, respectively. As the number of patients is the same for all SNPs, the number in the entry ”10” of the second-order contingency table, is equal to the sum of the entries ”100”, ”101”, ”102” of the third-order table, this relation is defined by Equation 3.4. As the vector G2 of the last SNP is neither streamed nor generated in the CTU, the reconstruction units implement Equation 3.5.

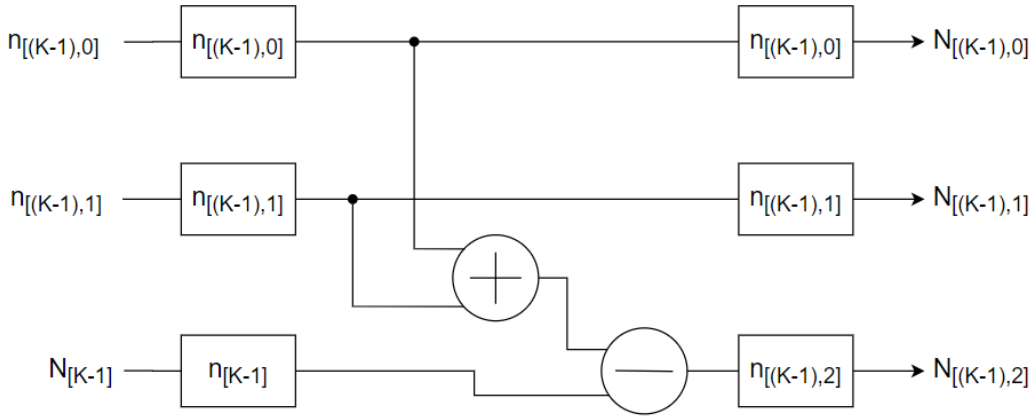

Figure 3.5: Basic Reconstruction Unit Datapath for cases

Figure 3.5 shows half of the datapath of a reconstruction unit. Per clock cycle, it receives one value of the vectors $n_{[(K-1),0]}$, $n_{[(K-1),1]}$, corresponding to the entries of the desired contingency table that depend on G0 and G1 of the last SNP of the combination, respectively, and one value of $n_{(K-1)}$. The output of the reconstruction unit is one value of the vectors $n_{[(K-1),0]}$, $n_{[(K-1),1]}$ and $n_{[(K-1),2]}$ (per clock cycle), corresponding the complete contingency table.

Each of the described reconstruction units are able to process $1/(3^{K-1})$ of a contingency table every clock cycle. Hence, the number of reconstruction units that is implemented for each CTU depends on the way that the partial contingency tables are sent from the CTUs. This process, is dependent on the number of patients in the dataset an on the order of interactions being tested.

3.2.4 Partial Table Transfer

The partial tables that are created in the accelerator’s CTUs, are sent to the reconstruction units, where their missing values are calculated. This section details the process of sending the partial contingency tables to the reconstruction units, which is done differently depending on the number of patients in the targeted dataset. This happens because the values that are generated in each CTU must be sent to the reconstruction units before the next values are generated. The number of clock cycles between the generation of two complete sets of values in the CTUs depends on the number of patients. As such,

for more patients, there are more available clock cycles for the transfer and reconstruction of the partial tables, which creates an opportunity for sharing the reconstruction units.

The transfer of the values stored in the CTUs and the reconstruction of the missing values, both for cases and controls is done in parallel and in the same manner. Accordingly, this section will be solely focused on half (either cases of controls) of the CTUs and the reconstruction units. Naturally, the number of input and output ports, and the number of values that are sent from the CTU to the reconstruction units corresponds to a single half of the contingency tables.

The partial contingency tables that are created in the CTUs are sent to the reconstruction units using three output ports. Those ports correspond to the three input ports of the reconstruction units (shown in Figure 3.5). Despite the tables not being complete, the number of values that are sent is equal to the number of entries of the contingency table, which depends solely on the order of interactions, and it is given by 3^K. From those values, two-thirds are values of the contingency table of K^{th}-order, while the other third is the complete contingency table of $(K-1)^{th}$-order. The latter is necessary for the calculation of the missing values of the table.

Each of the three ports is responsible for the transfer of $3^{(K-1)}$ values to the reconstruction units. However, the number of clock cycles that can be used to transfer that many values varies with the number of words necessary to stream an entire SNP through the communication interface, which, depends on the number of patients (as detailed in Section 3.1.2). This dependency exists because the number of clock cycles between the generation of two contingency tables in the same CTU is equal to the number of words per SNP. Also, every partial contingency table that is created in a CTU must be sent to the reconstruction unit before the next partial table is created. Consequently, for a higher number of patients, the transfer of the partial tables can take more clock cycles. Therefore, the number of clock cycles to send the values from a CTU to its respective reconstruction unit is the largest value that is divisible by $3^{(K-1)}$ and that is also smaller or equal to the number of words per SNP. Having more clock cycles to send the values to the reconstruction unit means that less values are sent in each clock cycle. As such, the number of values to be sent per clock cycle though each port is given by $3^{(K-1)}$ divided by the number of clock cycles used to send the values.

As an example, if the number of patient words is less than three, since the number of words is always even, this means that two clock cycles after a partial contingency table is created, another table is created and must be stored. Therefore, the stored tables need to be sent to the reconstruction unit in two clock cycles. However, since two is not divisible by $3^{(K-1)}$, the table is sent in just one clock cycle. To do so, each port sends the totality of the $3^{(K-1)}$ values per clock cycle. However, if the number of words per SNP is larger or equal to three and less than nine, the transfer of the values can take three clock cycles. This results in each port sending $3^{(K-2)}$ values per clock cycle during those three clock cycles. This pattern is repeated until the number of words per SNP is larger than $3^{(K-1)}$. In that situation, each port only sends one value per clock cycle during $3^{(K-1)}$ clock cycles.

The number of reconstruction units that are implemented for each CTU depends on the number of values per clock cycle that are sent to the reconstruction unit, and on the number of patients words necessary to stream on entire SNP from the on-board memory. Hence, the total number of reconstruction

units to be implemented in a design is given by the product of the number of reconstruction blocks, which is a group of reconstruction units, with the number of reconstruction units in each block. The number of reconstruction blocks is given by

$$\#Rec\_Blocks = \left\lceil \frac{\#Units}{\lfloor \#Words\_SNP / \#Cycles \rfloor} \right\rceil, \tag{3.6}$$

where $\#Cycles$ represents the number of cycles needed to send the partial contingency table from the CTUs to the reconstruction block. $\#Words\_SNP$, given by Equation 3.1, is the number of words needed to stream one entire SNP from the on-board memory to the FPGA, and depends on the number of patients and on the width of the data communication interface.

Every value that is sent in one clock cycle is processed in one clock cycle, as each unit accepts one value per clock cycle in each port. Therefore, the number of required reconstruction units in each reconstitution block is equal to the number of values that each port sends in one clock cycle, which is given by Equation 3.7

$$\#Rec\_Units\_per\_Block = \begin{cases} \frac{3^{K-1}}{\#Cycles}, & \#Cycles < 3^{K-1} \\ 1, & \#Cycles \geq 3^{K-1} \end{cases} \tag{3.7}$$

In the worst case, when the number of patients only requires two words to stream an entire SNP through the communication interface, only one cycle is necessary to sent the table. Therefore, the number of reconstruction blocks is one half of the number of units, as each block can be shared between two units, and $3^{(K-1)}$ reconstruction units are implemented in each reconstruction block. However, when the number of words per SNP is higher than $3^{(K-1)}$, only one reconstruction unit is implemented in each reconstruction block, that can, in turn, be shared among several CTUs. In this situation, the number of reconstruction units that are implemented is given by Equation 3.8, where the divisor of the first fraction is the number of CTUs that share one reconstruction block.

$$N\_rec\_units = \left\lceil \frac{N\_units}{\left\lfloor \frac{N\_words}{3^{(K-1)}} \right\rfloor} \right\rceil \tag{3.8}$$

A key takeaway of the process of transferring the partial contingency tables to the reconstruction units is that the more patients the dataset contains, the fewer reconstruction units are implemented, as the transfer of the tables can span across more clock cycles, allowing for the sharing of the reconstruction units. Also, as explained in Section 3.3.2, the number of implemented Mutual Information Units (MIUs) is proportional to the number of reconstruction unit. As such, having more patients also allows for the implementation of less reconstruction units and MIUs, which, in turn, allows for the implementation of more CTUs, increasing the performance of the generated architecture. Additionally, a higher number of patients allows using less wide output ports of the CTUs, as the number of values to send in each clock cycle decreases. Increasing the number of patients further results in an increase of the width of the ports, as the necessary number of bits to represent one contingency table entry also increases.

3.3 Mutual Information Calculation

As mentioned in the beginning of the chapter, the objective function used to classify the combinations is the mutual information. As described in Section 2.2.4, mutual information can be defined according to:

$$I(X;Y) = \frac{N[H(X) - H(X,Y)]}{N} - H(Y), \tag{3.9}$$

$$N[H(X) - H(X,Y)] = \sum [n_0 * \log_2(n_0) + n_1 * \log_2(n_1) - (n_0 + n_1) * \log_2(n_0 + n_1)] \tag{3.10}$$

where N (number of patients in the dataset) and $H(Y)$ are constants that only depend on the dataset, and n_0 and n_1 represent entries of the contingency table for cases and controls, respectively.

As such, only $N[H(X) - H(X,Y)]$is calculated for every contingency table, since that value is sufficient to sort the combinations by their mutual information value. This allows saving the resources needed to implement a floating-point division and subtraction in each MIU. The complete mutual information value is latter calculated in software only for the combinations that are saved.

Upon the calculation of the mutual information score, the values are sent to a unit that saves the best X values, as is illustrated in the Figure 3.1. The unit not only saves the best value (or a set of best values), but it also keeps track of the combination that corresponds to those values and stores that information along with the $N[H(X) - H(X,Y)]$ values. Accordingly, when all the combinations are tested, the save units send their saved values to the host, where they are divided by the number of patients in the dataset and the value of $H(Y)$ is subtracted. The values are then sorted and the X combinations with the highest mutual information value are reported.

3.3.1 Mutual Information Unit

Figure 3.6 details the architecture of each MIU, responsible for the calculation of $N[H(X,Y) - H(X,Y)]$. The unit accepts a pair (comprising a case and a control value), of entries from the complete contingency tables, sent by the reconstruction units every clock cycle and returns the corresponding partial $N[H(X,Y) - H(X,Y)]$ value.

To save resources that would otherwise be needed to implement the calculation of $n * log_2(n)$ in hardware, a look-up table is used. Hence, instead of calculating the value, all the possible results are stored in a BRAM using a single-precision floating-point representation (in the address corresponding to the input n, the BRAM contains the single-precision floating-point representation of $n * log_2(n)$). The necessary range of values of the look-up table depends on the number of patients in the dataset. As n is an entry of a contingency table, its maximum value is, for a balanced dataset, half of the number of patients. Therefore, the number of entries of the look-up table in the top half of Figure 3.6 is equal to half of the number of patients. Consequently, since the input of the second table is the sum of two contingency table entries, the maximum value of its entries is the total number of patients.

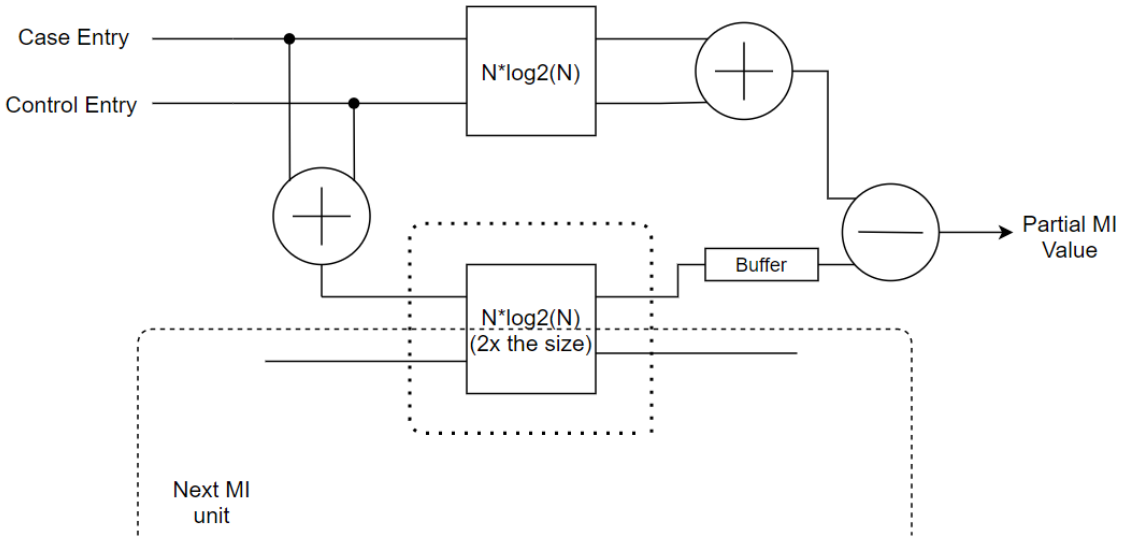

Figure 3.6: Mutual information unit overview

The adder that sums both entries of the contingency table ($[n_{Case} + n_{Control}]$), and the adder that sums the outputs of the look-up table in the upper arm of the datapath ($[n_{Case} * log_2(n_{Case}) + n_{Case} * log_2(n_{Case})]$) have different latencies. This happens because the first is adding two natural numbers, whereas the second is adding two single-precision floating-point values, which is a more complex operation with higher latency. As such, the inputs of the subtractor are not available at the same time. To solve this issue, a buffer is used in the bottom part of the datapath to introduce as many clock cycles of latency as the difference between the latencies of the two adders. Further details regarding the implementation of the look-up tables for the calculation of $n * log_2(n)$ and the implementations of the floating-point operation, namely the addiction and subtraction, are presented in Section 4.3.

3.3.2 Number of Mutual Information Units and Mutual Information Values

The number of MIUs used in the proposed architecture is directly correlated with the number of reconstruction units that are implemented, and both depend on the number of patents in the targeted dataset and on the value of K. As the MIUs do not calculate the final mutual information score (only one partial value for each case/control pair), more than one MIUs are needed for the complete calculation of the mutual information score of a contingency table. This Section details the amount of necessary MIUs and how the values are added and accumulated to produce the final mutual information value.

As it is detailed in Section 3.2.3, the reconstruction unit created using the general method for any K (represented by Figure 3.5), can output six values of the complete contingency table every clock cycle (three for half of the table corresponding to cases and the other three for the half of the table corresponding to the controls). Therefore, as the MIUs accept two contingency table entries, one for cases the other for controls, per clock cycle, each reconstruction unit is connected to three MIUs, so that each of the three pairs of values that are outputted by the reconstruction units can be processed in parallel.

Since each MIU calculates a partial mutual information value, the values that are outputted from the units in the same clock cycle are added. When the partial contingency tables are sent from the CTUs to the reconstruction units in more than one clock cycle, the number of reconstruction units decreases, and, consequently, so does the number of MIUs. In that case, the calculation of the partial mutual information values span across as many clock cycles as the transfer of the tables. As such, the values produced by the MIUs in the same clock cycle are added and then accumulated with the values produced in the next clock cycles, until all the final mutual information values of that contingency table are produced. In short, the partial mutual information values calculated in the same clock cycle are added following a tree structure. The results of this sum then accumulated as many times as the number of clock cycles that are used to send the partial contingency tables from the CTUs to the reconstruction units. This accumulation is done inside of the save unit.

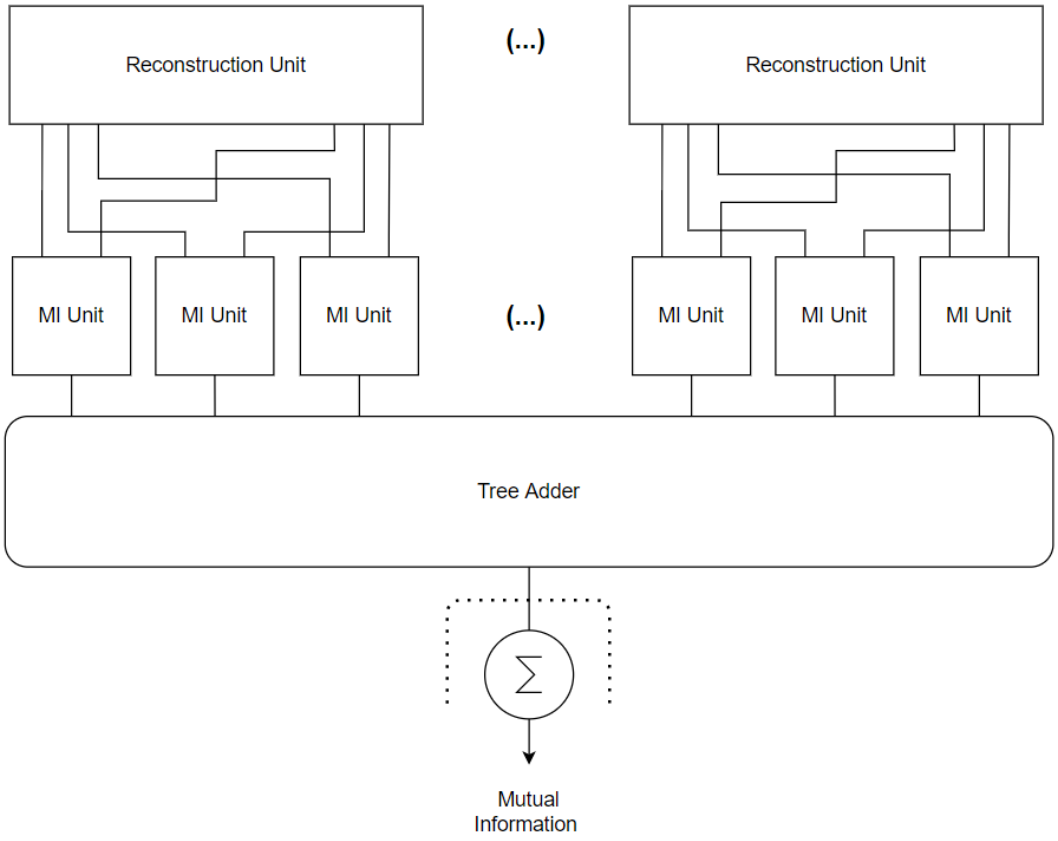

Figure 3.7: Mutual Information calculation

Figure 3.7 illustrates the schematic of the calculation of the mutual information for one contingency table. In a scenario where the number of patients is high enough that each one of the three ports that transfer the partial contingency tables only send one value per clock cycle, only one reconstruction unit is connected to the CTU. In that case, three MIUs are implemented. The partial mutual information values that are calculated in each clock cycle are added, and accumulated with the ones produced in the next $3^{(K-1)} - 1$ clock cycles.

On the other hand, the worst-case scenario is when the partial contingency tables are sent from the CTUs to the reconstruction units in one clock cycle. In that case, the number of implemented MIUs for each CTU is given by $3 * (3^{(K-1)})$, corresponding to three times the number of reconstruction units. All the partial mutual information values are, in this case, calculated in parallel without any sharing of resources. Therefore, all partial values are added in a three structure, which outputs the final mutual information value, as all the partial values have already been accumulated. In this situation, there is no need for an accumulator.

3.3.3 Compare and Save Mutual Information Values

As described before, the partial values that are calculated in the MIUs are sent to the save unit, where they are accumulated as many times as the number of cycles that it takes to stream an entire contingency table, except in the particular situation where the contingency table must be streamed in only one clock cycle. In that case, the value that is received by the save unit is the complete $N[H(X) - H(X,Y)]$.

The complete value of $N[H(X) - H(X,Y)]$ is compared to the previously stored value. The highest $N[H(X) - H(X,Y)]$ value is stored while the others are discarded. This approach guarantees that the highest mutual information value, and its respective combination, is stored in one of the save value units. However, despite sending to the PS more than one combination, only one of those corresponds to the highest mutual information value globally, as they are only compared between combinations generated in CTUs that share one *save value* unit.

Specifically, the values that are stored in each save unit are local maximums (of the CTUs that share one *save value* unit) and not global maximums. Therefore, only the best value is guaranteed to be stored in one of the units, as the second best combination could have been produced by the same unit that produces the best combination. To ensure that a set of the X best combinations are sent to the host, X mutual information values, and $(X * K)$ SNP numbers (to identify the combination), have to be stored in each *save value* unit. This requires the implementation of X comparators and X sets of registers to store the $N[H(X) - H(X,Y)]$ values and the combination identifiers in each unit.

For every new complete value of $N[H(X) - H(X,Y)]$ that is produced in the *save value* unit (or upon the end of a round), and depending on the re-initialization mode that is used, the counters that keep track of the combinations that are being tested are updated according to the process described in Section 3.2.1.

Once the processing of all combinations is finished, all implemented save units sequentially send the mutual information values and the identifiers of the corresponding combination to the Host, where they are stored and filtered, through the same interface that streams the dataset from the on-board memory to the FPGA.

3.4 Architecture Optimizations for Second-Order Interactions

Although the general architecture proposed in this Chapter enables the creation of FPGA-based accelerators for any order of combinations, it is possible to take advantage of the smaller contingency tables that are created for second-order interactions to introduce some optimizations that are specific for this case. However, these optimizations do not apply to the general architecture due to the added complexity in the reconstruction of the contingency tables. This would invalidate the scalability of the design with the order of interactions, as a dedicated reconstruction unit would have to be designed for each specific order of interactions.

To apply the specific optimizations for second-order interactions, all the components of the architecture (except the mutual information and the *save value* units), are modified, along with the method to transfer of the partial CTUs. The changes made to each of the components are detailed in the following sections.

3.4.1 Contingency Table Units Architecture (K=2)

Figure 3.8 represents half of the datapath of the CTUs for second-order interactions, where some optimizations for this particular case are done. Each unit has, as expected, $2 * (K - 1) = 2$ pairs of BRAMs to store the vectors of zeros and ones of the stored SNP for both cases and controls. As such, all the units in the systolic array are equivalent. The optimized architecture only utilized four sets of logic AND ports, PopCount units, and adders. In opposition to the $2 * (3^{K-1}) = 6$ of each set that are needed for the general method. Also no NOR logic gates are used, as, with such optimizations, the G2 of the stored SNP is not required, as the spatially designed reconstruction units are able to calculate the entries of the table that depend on that vector. The simplified design for the CTUs for second-order epistasis detection, also allows for the reduction of the number of pipeline stages from five to four without impacting the timing requirements.

When applying these optimizations, the *count last* unit is also changed to produce the PopCount values of the vectors of zeros and ones for all SNPs that are streamed, using two PopCount units (one for cases and one for controls), and send those values to the systolic array of CTUs.

As detailed in Section 3.2.3, with the PopCount values that are produced in the *count last* unit, only the contingency table entries that do not depend on G2 of the SNPs that compose the combination are needed to reconstruct the entire contingency table in the reconstruction units. Therefore, each CTU generated using the specific optimizations for second-order interactions, produces only four values of the contingency table for cases and four for controls, instead of the six values for each that are produced if the general architecture is applied. This results from the fact that when using the general architecture, only the entries that depend on G2 of the last SNP of the combinations are not generated.

The process of generating the entries of the contingency table is similar to the one employed in the general implementation except for the first step, where the values of the PopCount of the $(K - 1)^{th}$ combination are calculated. That operation is not performed in the optimized second-order architectures, as those values are calculated in the *count last* unit and propagated through the systolic array. Similarly

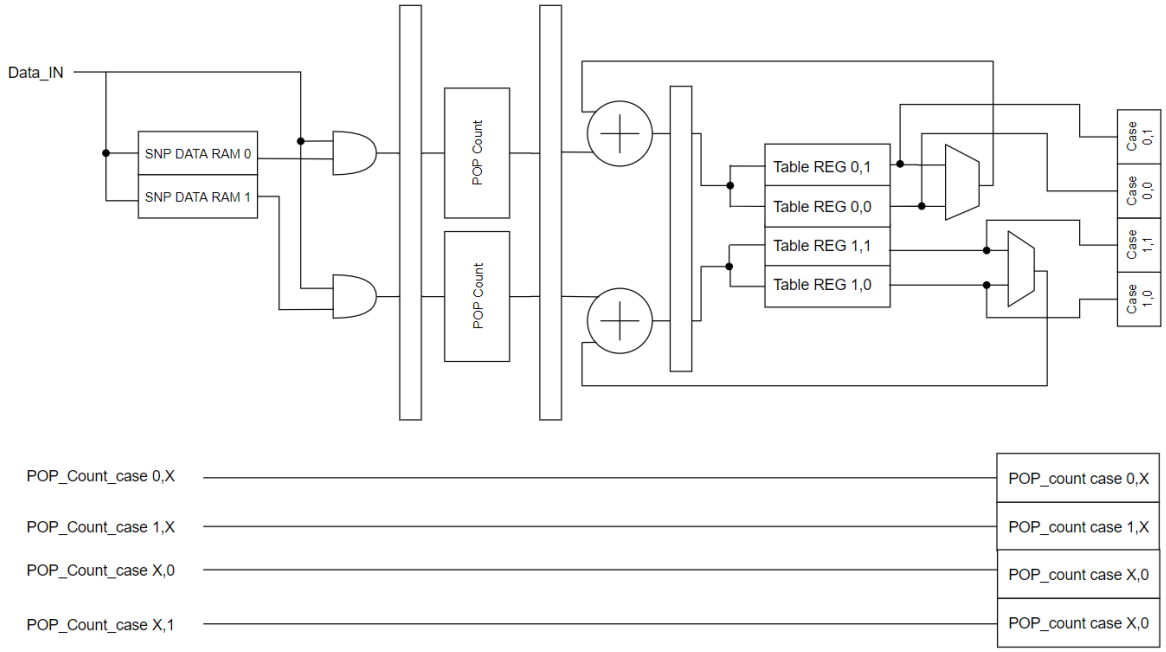

Figure 3.8: Datapath for half of the K =2 contingency table units

to the general implementation, when the last partial values have been accumulated and are stored in the *Table REG* registers, they are, sent to a second set of registers that store the partial contingency table until it is sent to the reconstruction unit. Although the throughput of the unit is the same as the units generated with the general method, since the number of pipeline stages for this second-order implementation was reduced from five to four, the latency of the unit is also reduced by the same factor.

3.4.2 Reconstruction Units Architecture (K=2)

Due to the difference in the number of values that are produced by the CTUs in the general implementation, when using the optimized design to process second-order interactions, the method used for reconstruction of the missing contingency table values is fundamentally different. Accordingly, a dedicated architecture is necessary to calculate the missing values for second-order epistasis detection.

In particular, in this specially devised architecture, G2 of the stored SNP is never generated, as opposed to what is done in the general method. Therefore, the contingency table entries that depend on G2 of any of the two SNPs of the combination are not generated. As such, only four values for cases and four values for controls are generated for each contingency table.

The missing values of the contingency table are calculated as per the Equation 3.11, where $n_{(0,X)}$, $n_{(1,X)}$, and $n_{(2,X)}$ represent the PopCount for the first SNP of the combination, of G0, G1, and G2, respectively. $n_{(X,0)}$, and $n_{(X,1)}$ represent the PopCount for the second SNP of the combination, of G0 and G1, respectively. Since G2 is never generated, the value if $n_{(2,X)}$ is calculated in the reconstruction units according to Equation 3.12, where N is the total number of patients in the cases or the controls group.

$$
\begin{cases}
n_{(0,2)} = n_{(0,X)} - (n_{(0,0)} + n_{(0,1)}) \\
n_{(1,2)} = n_{(1,X)} - (n_{(1,0)} + n_{(1,1)}) \\
n_{(2,0)} = n_{(X,0)} - (n_{(0,0)} + n_{(1,0)}) \\
n_{(2,1)} = n_{(X,1)} - (n_{(0,1)} + n_{(1,1)}) \\
n_{(2,2)} = n_{(2,X)} - (n_{(2,0)} + n_{(2,1)})
\end{cases} \tag{3.11}
$$

$$
n_{(2,X)} = N - (n_{(0,X)} + n_{(1,X)}) \tag{3.12}
$$

Figure 3.9 illustrates half of the reconstruction unit that is used to calculate the missing values of the contingency tables by following Equations 3.11 and 3.12. Contrary to the reconstruction unit for any order of interactions (that accepts and outputs three value per clock cycle), as it is fully pipelined, the reconstruction unit for second-order interactions accepts eight values for every nine clock cycles, this results from the fact that the unit outputs nine values, corresponding to the entire contingency table (sent to the MIUs at a rate of one value per clock cycle), and only receives eight values (at the same rate).

Similarly to the architectures that are generated using the general method, the number of reconstruction units that are implemented depends on the number of patients that the dataset contains. Details regarding the transfer of the partial contingency tables and the number of reconstruction units that are implemented when implementing the specific optimizations for second-order epistasis detection, are presented in Section 3.4.3.

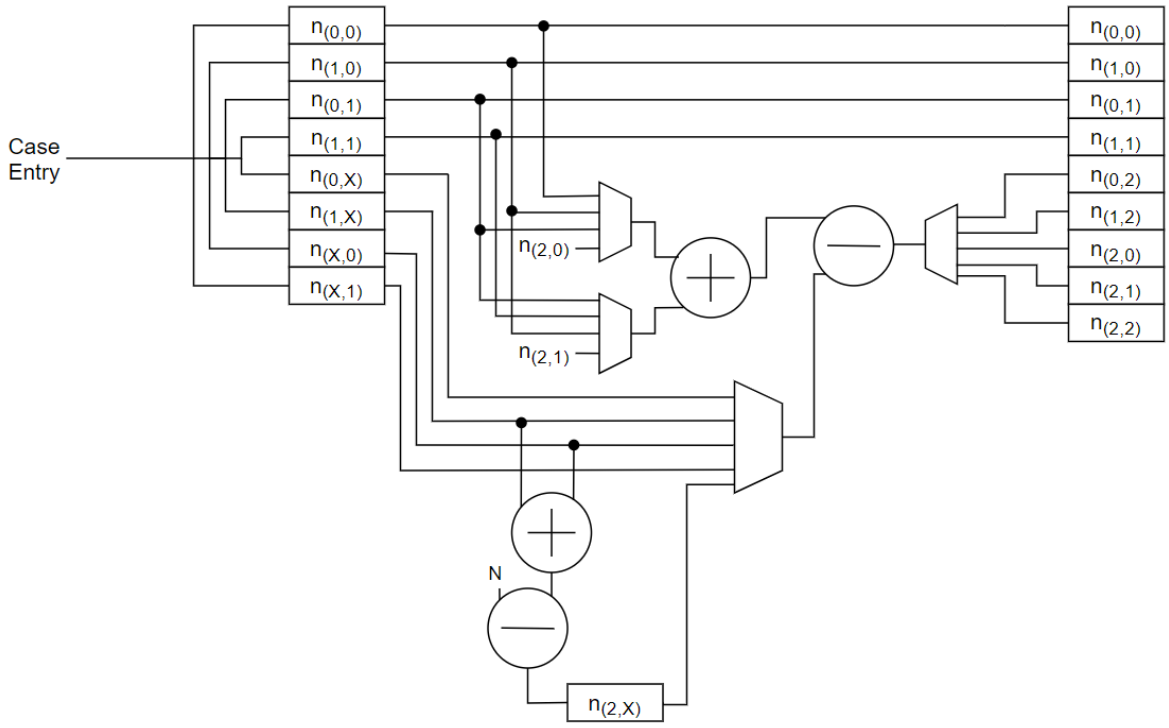

Figure 3.9: Reconstruction Unit Datapath for $K = 2$ for cases

3.4.3 Partial Tables Transfer (K=2)

The fact that the reconstruction of the missing contingency table entries is done differently for second-order interactions, results in a different number of required reconstruction units. The process used to send the partial contingency tables to the reconstruction units is different from the one used in the architectures generated using the general method. This is not only because the reconstruction units accept values differently but also because it takes advantage of the smaller contingency tables.

In particular, in the optimized architecture for second-order epistasis detection the number of values that are sent from the CTUs to the reconstruction units is eight. However, as the reconstruction units only accept eight values every nine clock cycles, those eight values are sent through a single bus in eight clock cycles and a stall in introduced afterwards. Therefore, when the number of words that are streamed through the data communication interface to represent one entire SNP is higher than the number of clock cycles that is needed to stream the partial contingency table (nine), the number of reconstruction units that are implemented is given by Equation 3.13.

$$N\_rec\_units = \left\lceil \frac{N\_units}{\lfloor \frac{N\_words}{9} \rfloor} \right\rceil \tag{3.13}$$

This equation is applicable for more than nine words per SNP, which is the same to say that it is applicable for more than eight words per SNP, as the number of words that define one SNP is always even. This means that this solution to transfer the partial contingency tables is only valid for more than $(R * 8)/2$ patients, where R is the width of the communication interface.

As the minimum number of patients required for this solution to work is much lower when processing second-order interactions than for higher-order, when the number of patients is lower than the limit, dummy patients are added to the dataset to ensure that at least ten words per each SNP are sent to the FPGA. Particularly, this solution impacts the performance of the architecture, as the execution time for a dataset with any number of patients below the $(R * 10)/2$ is the same. While, a solution that works for every number of patients (similar to what is proposed in the general implementation) could be used, that would require more output ports in the CTUs to send the partial contingency tables faster, which would increase the complexity of the design. In fact, reducing the number of contingency tables that could be implemented for a higher number of patients. Although this solution would increase the performance for a number of patients below $(R * 10)/2$, it would reduce the performance of the architecture for every number of patients higher than $(R * 10)/2$.

Accordingly, the first solution is only applicable for the second-order case, as the minimum number of patients for which no dummy patient is added (given by $R * (3^K + 1)/2$) is much lower for second-order epistasis detection, than for higher-orders of interactions. The number of dummy patients that is added to the dataset in the second-order solution is given by Equation 3.14.

$$N\_dummy = R * (3^K + 1)/2 - N\_patients \tag{3.14}$$

Similarly to the general method for any order of interactions, in this particular case, the more patients

are present in the dataset, the more resources can be shared, as the number of CTUs that share a reconstruction unit is proportional to the number of patients. Also, when implementing the particular optimizations for processing second-order interaction, the number of MIUs is the same as the number of reconstruction units (as detailed in Section 3.3.2). Consequently, a higher number of patients results in the implementation of less reconstruction and MIUs, allowing resources to be freed implement more CTUs, resulting on for the processing of more combinations in parallel.

3.4.4 Mutual Information Calculation (K=2)

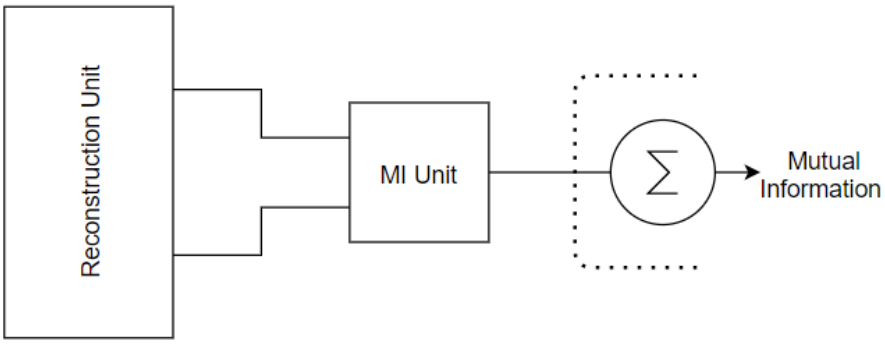

Figure 3.10: Mutual Information calculation for $K = 2$

As detailed in Section 3.4.3, when using the specific optimization for second-order interactions, one CTU, or group of CTUs, is only connected to one reconstruction unit, which outputs two values per clock cycle (corresponding to one case and one control pair of a complete contingency table). As such, only one MIU is implemented for each reconstruction unit. The MIU produces one of the nine partial mutual information values in each clock cycle, that, similarly to the general method for any order of interactions, are accumulated in the *save value* unit.

The process used for the mutual information calculation in the second-order (illustrated in Figure 3.10), is simpler and uses fewer resources, than the designs that are generated when using the general method for any order of interactions. This is because it is not necessary to add a variable number of partial mutual information values that are calculated in the same clock cycle, and only one MIU is implemented. Such an approach is made possible by taking advantage of the smaller contingency tables to simplify their transfer and, consequently, simplifying the mutual information calculation.

3.5 Summary

This Chapter detailed the specialized accelerator architecture for K^{th}-order epistasis detection proposed in this thesis. The proposed architecture is capable of being parameterized to generate efficient hardware implementations to perform epistasis detection of any order of complexity, targeting datasets

with any number of patients. The proposed method is described generally in regards to the order of the interactions, to the number of patients and the bandwidth between the memory and the FPGA.

The Chapter addressed the creation of the units that are responsible for creating the contingency tables for any order of interactions, as well as the efficient transfer of the partial contingency tables to the unit that reconstructs them, sharing as many resources as possible. The CTUs, as well as the table communication subsystem, are particularly dimensioned to the number of patients in the dataset. This allows the proposed method to be applied to any number of patients, contrarily to other existent specialized hardware architectures. The calculation of the mutual information value for all the evaluated combinations is also described.

In the last section, an optimized architecture for second-order epistasis detection (taking advantage of the smaller contingency tables for second-order interactions) is proposed. The section details the alterations made to the general architecture to implement the proposed optimizations.

Chapter 4

Accelerator FPGA Implementation

This chapter details the implementation of several architectures generated using the method described in Chapter 3, by varying the parameters used to generate architectures targeting different orders of interactions (second, third, and fourth) and different numbers of patients. For each design, the number and organization of the functional blocks (depending on the number of CTUs) are detailed.

Although the hardware designs for high-order epistasis detection that are generated according to the generic architecture proposed in Chapter 3 can be implemented in any FPGA, as long as the device in question has enough resources to implement at least one CTU, other requirements (specified in Section 4.1) must be met so that the complete application can be executed in the same board.

4.1 Implementation Platform Details

As is was defined in Chapter 4, the proposed accelerator architecture was specifically designed to target computing systems composed of an FPGA device connected to a host CPU, via an high-performance data communication interconnection. Accordingly, in the scope of this thesis, the proposed architecture was implemented by targeting the Xilinx Zynq SoC family of FPGA platforms. These platforms are composed by a tightly-coupled CPU-FPGA system, comprising an ARM CPU directly connected to a Xilinx FPGA fabric though an AXI4 interface (further details in Section 4.2.2).

As the AXI4 interface, present in this family of SoCs, allows for an high-bandwidth connection between the FPGA fabric the CPU and the on-board memory, the required system can be fully implemented in such a SoC. As the system is tightly-coupled inside the same SoC, it provides very low power consumption as no other external device has to be powered, without compromising the performance of the system.

4.2 Targeted Boards overview

The board where most of the generated architectures were implemented, is a ZYNQ-7 Mini-ITX Motherboard [58], equipped with a Zynq XC7Z100 SoC [59]. However, the ZYNQ-7 Mini-ITX is not the

only board that can run the complete application. Every board that features a Zynq SoC is suitable for the implementation of the proposed architectures. As such, a Xilinx Zynq UltraScale+ ZCU102 board [60] featuring a ZU9EG Zynq SoC [61] is also used to implement some of the generated architectures, as a way to extend further the levels of performance.

Details regarding the features of both boards are given below, where the general organization of a Zynq SoC is explained along with some of the particular features of the Zynq-7000 and the Zynq-Ultrascale+ SoCs contained in the ZYNQ-7 Mini-ITX and Zynq UltraScale+ ZCU102 boards, respectively.

4.2.1 Zynq SoC Device Family

Zynq is the name given to a family of SoCs developed by Xilinx designed for hardware/software co-design. A Zynq SoC is composed of a Processing System (PS) responsible for running software, and Programmable Logic (PL) consisting of FPGA fabric that can be used to implement any logic circuit as long as it fits in the FPGA, allowing for the easy deployment of custom designed accelerators to work alongside the PS that can be used to accelerate workloads. The PS and PL are connected via a series of AXI4 interfaces that provide high bandwidth access to the on-board memory. The Zynq platform provides a great amount of flexibility, not only because of the reconfigurable logic but also because of the number of components that can be included in the board in which they are integrated. This makes Zynq boards suitable for applications such as the automotive industry, defense, aerospace, multimedia, and others. Figure 4.1 shows a block diagram of some of the components contained in a Zynq SoC, detailing the organization and connection between the PS and PL.

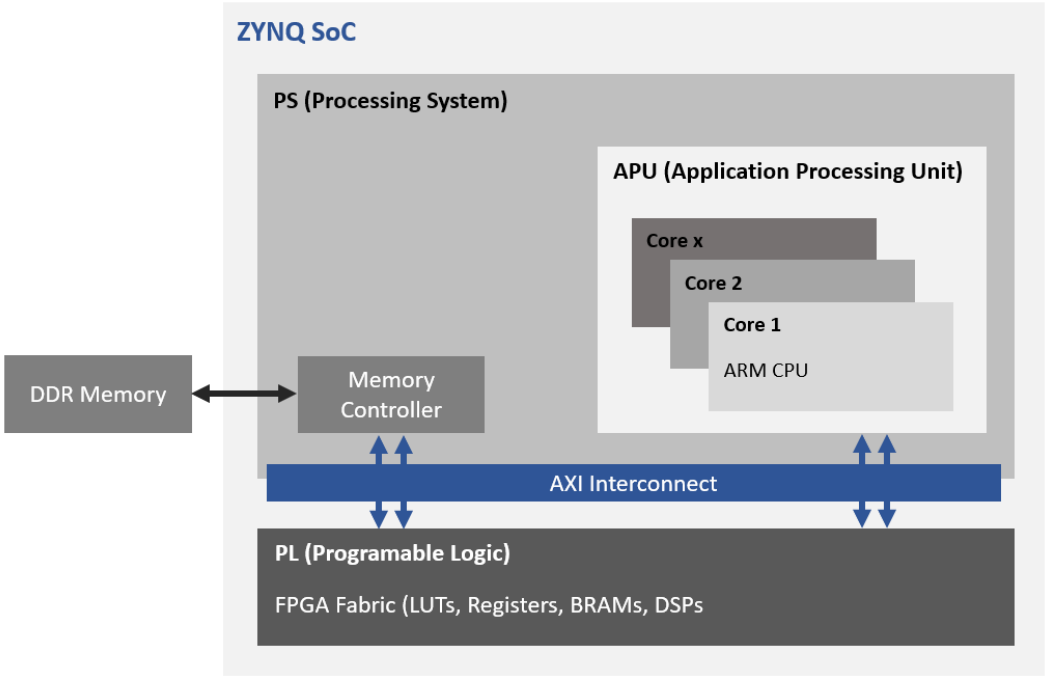

Figure 4.1: Zynq Ultrascale+ MPSoC Block Diagram

4.2.2 AXI4 interface

The AXI protocol, that is used for data transfer between the board and the PS and the PL, is a high-performance, multi-master, multi-slave synchronous interface designed for on-chip communication, included in ARM Advanced Multicontroller Bus Architecture (AMBA). The AXI4 interface works with a basic handshake protocol between two signals, XVALID, driven by the source informs the destination whether the data on the channel is valid and ready to be read or not, and XREADY, driven by the destination informs the source whether it is ready to receive data. When those two signals are high in the same clock cycle, a transfer occurs. Once a source changes the XVALID signal to high, it has to stay high until a handshake occurs. As both the source and the destination can regulate the data flow, either one of them can throttle the speed of the transfer.

There are 3 different types of the AXI4 interface:

- AXI4: Memory-mapped interface with support for bursts of up to 256 data words (also named beats), Quality of Service signaling and parameterizable Data Width.
- AXI4-lite: Subset of the AXIs interface with simpler control and smaller footprint. It Does not support for bursts and all accesses are of the same size of the bus.
- AXI4-Stream: Unidirectional data streams from master to slave with unlimited burst length.

All 3 types of the AXI4 interface use the previously described handshake to initiate data transfers and only differ on the number of used channels and the type and amount of data that can be transferred. In this thesis, the adopted interface is the AXI4, also known as AXI4-FULL. The interface is constituted by 5 channels:

- Write Address channel (AW) - Provides the address and control information for the data to be written.
- Write Data channel (W) - Transfers data from the master to the slave.
- Write Response channel (B) - Response from the salve to the master after a write transaction is completed.
- Read Address channel (AR) - analogous to the write address channel for reading.
- Read Data channel (R) - Transfers data from the salve to the master.

The full AXI4 interface is also used in the proposed architectures, due to the support for bursts and the ability to send data to and from the FPGA and the fact that it is memory-mapped. An AXI4 burst allows for multiple data transfers to occur with a single request, AXI4 support three modes of burst, FIXED, INCR, and WRAP, regarding the memory address where each beat starts.

In the FIXED mode the start address for each beat of that burst in always the same, in the INCR mode, the start address of the beat is equal to the last address plus the number of words that constitute the beat, as such, the memory addresses in use are sequential within the burst. This type of burst is the one used in the proposed architectures to stream the dataset to the FPGA, as the data to send in one

round is always contguous in memory. The WRAP burst mode is similar to the INCR but only operates in a specific memory range, which means that once the upper range of memory is reached, the start memory address of the next beat will be the lower in the specified range.

Zynq-Ultrascale+ SoC

The PS of a Zynq-Ultracale+ SoC (such as the one present on the Xilinx Zynq UltraScale+ ZCU102 board), contains an application processing unit featuring a quad-core ARM Cortex-A53 64-bit processor implementing the ARMv8-A instruction set architecture with a working frequency of up to 1.5GHz. The level 1 cache has 32KB, separated in instruction and data, and is independent for each core. An 1MB 16-way associative 2nd level cache is unified for all cores and supports Error Correcting-Code (ECC). The PS also features a Real-time Processing Unit containing a dual-core ARM Cortex-R5, Full-Power and Low-Power domain Direct Memory Access (DMA), a GPU and dynamic memory controller.

The processing system is connected to a 4GB/64-bit memory module with ECC and has support for high-speed connectivity peripherals.

The Zynq Soc contains several AMBA AXI4 interfaces between the PS and the PL, including slave and master High-Performance (HP) interfaces from PL to PS and interfaces from the PL to the Cortex-A53 cache memory. Those interfaces can be configured with 128, 64 or 32-bits data widths. The six HP AXI4 ports allow for high bandwidth data transfers from the PL to the DDR and the high-speed interconnect in the PS, the connection between the memory and the PL makes use of a 1KB FIFO.

The PL contained in this device is connected to a 512MB/16bit DDR4 memory accessible via a Memory Interface Generator, and can also access the 4GB DD4 memory connected to the PS using the AXI interface. The FPGA uses the same fabric as the Virtex and Kintex UltraSCALE+ devices based on 16nm technology. An overview of the FPGA resources in the present in the PL are stated in Table 4.1. The block RAMs contained in this FPGA can store up to 36Kbit of data, either configured as one RAM or as two independent 18Kbits RAMs.

Table 4.1: Zynq Ultrascale+ SoC FPGA resources

System Logic Cells	CLB Flip-Flops	CLB LUTs	Block RAMs	DSP Slices
599 550	548 160	274 080	912	2520

The ratio between the number of Look-Up Tables (LUTs) and the number of BRAMs in this FPGA, is 300 LUTs for each BRAM, and the ratio between the number of LUTs and the number of Digital Signal Processors (DSPs) is 109 LUTs for each DSP.

Zynq-7000 SoC

The PS of a Zynq-7000 SoC, present in the ZYNQ-7 Mini-ITX board, is composed of an application processing unit based in a dual-core ARM Cortex-A9, which implements the ARMv7-A instruction set architecture and reaches a maximum clock frequency of 1GHz. Several AMBA AXI4 interfaces between

the PS and the PL are available, which include two slave and two master AXI4 ports, and four HP slave AXI4 interfaces with direct access to the DDR memory, and configurable with a width of 32 or 64 bits.

The FPGA fabric in the Zynq-7000 SoC that is present in the ZYNQ-7 Mini-ITX Motherboard, based on a 28nm technology is connected to the PS by master and slave AXI4 interfaces. The four HP interfaces (which can be used for high-bandwidth data transfers between the PS and the PL) can be configured with a width of 32 or 64 bits. Table 4.2 details the number of available resources in the FPGA of the Zynq XC7Z100 SoC.

Table 4.2: Zynq SoC FPGA resources

System Logic Cells	CLB Flip-Flops	CLB LUTs	Block RAMs	DSP Slices
444 000	554 800	277 400	755	2020

The ratio between the number of LUTs and the number of BRAMs, relevant in the second-order implementations, is 365 LUTs for each BRAM, and the ratio between the number of LUTs and the number of DSPs is 137 LUTs for each DSP.

4.2.3 Other used FPGAs

The generated architectures can be implemented in any FPGA. However, if the FPGA is not integrated into a Zynq or a similar SoC, the complete system to perform the epistasis detection requires additional steps to be implemented that are not contemplated in this thesis. This is due to the fact that the system relies on software application, running on a Host CPU to control the memory addresses and the size of each data stream. Alternatively, a similar system can be built using a PCI express interface existent in the boards mentioned in this section to transfer data between the host machine memory and the FPGA, and run a similar software in the host machine CPU. However, implementation of such a system is out of the scope of this thesis.

As such, to obtain results when using an FPGA that is not integrated into a Zynq SoC, only the FPGA accelerator design is implemented, and it is assumed that the interface that would be used would perform in a similar manner as the AXI4 interface present in the Zynq SoCs.

Virtex-7

To better compare the generated architectures with the ones proposed in the state of the art, some of the generated architectures are implemented in an Alpha Delta ADM-PCIE-7V3 board, which contains a Virtex-7 690T FPGA, based on the same 28nm technology as the FPGA contained in the Zynq-7000 Soc. Table 4.3 details the number of relevant FPGA resources available in the Virtex-7 690T FPGA.

Table 4.3: Virtex-7 690T resources

System Logic Cells	CLB Flip-Flops	CLB LUTs	Block RAMs	DSP Slices
693 120	866 400	433 200	1470	3600

The ratio between the number of LUTs and the number of BRAMs in this FPGA, is of 295 LUTs for each BRAM, and the ratio between the number of LUTs and the number of DSPs is of 120 LUTs for each DSP, which is translates in lower number of DSPs, relative to the number of LUTs, when compered with the FPGAs in the Zynq SoC used.

4.3 Generic Implementation Details and Requirements

The hardware and software stacks present in a Zynq SoC and the high bandwidth connection to the on-board memory provided by the AXI4 interface makes it suitable for the implementation of the architectures generated with the proposed method. Although the logic designs can be implemented in any FPGA, the software responsible to calculate the memory addresses, the size of the data transfers to the FPGA, and to manage the AXI4 interface, runs in the ARM CPUs integrated on the SoC.

The following paragraphs provide details regarding the implementation of some of the operators used throughout the design, particularly in the CTUs and in the MIUs.

4.3.1 Contingency Table Creation

This section discusses the design choices regarding the bandwidth of the AXI4 interface, as well as implementation details of some of the elements contained in the CTUs. Particularly emphasis is given to the buffers that store the SNP data, the PopCount units, the adders, and to the width of the registers that store each entry of the contingency tables.

AXI4 Interface Width

Since the ZYNQ7 Mini-ITX board was targeted during the development of the proposed architecture, the width of the AXI4 interface (referred to as R throughout Chapter 3) is 64-bits, as it is the maximum supported width for the AXI4 interface in that board. Therefore (as explained in Section 3.1.2), in a balanced dataset (with the same number of cases and controls) such as the ones targeted, 32 bits are used to send data regarding the patients in the control group and the other 32 bits are used for the patients in the cases group.

Although more than one AX4 interface could be used to increase the data bandwidth between the on-board memory and the FPGA, it would also increase complexity of each CTU, as the AND ports, the PopCount units and the adder would have to be larger (since the words that go through them would be wider). Increasing the bandwidth also induces more strain to the transfer of the partial contingency tables to the reconstruction units, as the number of clock cycles to transfer the tables depends on the value R (detailed in Section 3.2.4). As such, it would result a negative impact in the performance of the architecture, as the number of CTUs that can be implemented would, in turn, decrease. Accordingly, to avoid the overhead of using more than one AXI4 port and of their synchronization, a single port is implemented with a width of 64-bits.

In the Zynq-Ultrascale+ ZCU102, the width of the AXI4 interface could be increased to 128-bits. Nevertheless, for the reasons stated above, less CTUs would be implemented, which would also lead to a decrease in performance when compared to the use of a 64-bits AXI4 interface.

SNP Storage

As detailed in Section 3.2.2, each CTU in the systolic array stores data corresponding to one complete SNP, with the exception of the first unit, which stores data corresponding to $K-1$ SNPs. Each SNP is stored in one unit using four BRAMs, one pair for the cases, and the other pair for controls. Also, each BRAM in a pair is used to store the G0 and the other to store the G1. Hence, as $R = 64$), each BRAM stores as many 32-bit words as half on the number of words needed to send one patient, in a balanced dataset, which is given by Equation 3.1 (defined in Section 3.1.2).

With the exception of the first unit of the systolic array, the number of SNPs that are stored in each CTU is the same for every order of combinations. As a result, the BRAM utilization of each CTU does not vary with the order of interactions, only with the number of patients. Since the BRAM utilization in each CTU depends on the total number of CTUs and in the number of patients, for a second-order implementation (where the number of implemented contingency tables is higher as each one is simpler), the number of required BRAMs often exceeds the total number of available BRAMs . When that situation occurs, the memory structures required to hold the SNP data are implemented as logic (i.e., distributed memory or registers).

PopCount Unit

As detailed in Section 3.2.2, each CTU contains $3^{(K-1)}$ PopCount units. A PopCount unit can be efficiently implemented by taking advantage of the 6-input LUTs available in modern Xilinx FPGAs, this is done by using each LUT to build a simple 6-bit hard-coded PopCounter, that returns a 3-bit vector.

Since the data words that go through the PopCount process are 32 bits wide, six of the described LUTs are used in the first stage of the PopCount unit. The six 3-bit vectors that result from the first set of LUTs could be added to produce the correct result. However, to minimize the resource utilization, those vectors are, instead, reorganized by their bit significance in three 6-bit vectors, where the first vector is constituted by the least significant bits of the six vectors that result from the first LUTs level, the second is constituted the second least significant bits, and the third vector by the most significant bits. The three generated 6-bit vectors go are passed to another three LUTs, outputting three 3-bit vectors (C0, C1, C2), which are added in accordance with their significance, as described in Equation 4.1.

$$pop\_count = (C0 * 2^0) + (C1 * 2^1) + (C2 * 2^2) \tag{4.1}$$

Adders

The values result from by the PopCount units correspond to partial contingency table entries. As such, to produce the final entries of the contingency table from their partial values, they are accumulated,

using the adders present in the CTUs. Those adders can be implemented either as logic (by a set of LUTs) or using the DSP48E1 slices [62] available in the Xilinx FPGAs. To efficiently use the resources of the board, some of the adders are implemented as logic while the rest of them are implemented with DSPs. In fact, the number of adders that are implemented in each way can be easily changed so that the recourse utilization of circuit can be better adjusted to the available resources of the targeted FPGA.

Furthermore, the ratio of adders that are implemented as logic or with DSP slices varies not only with the targeted board but also with the order of the epistasis detection being performed, and with the number of patients in the dataset. This is because (as explained in Section 3.3.2), the number of MIUs varies with the number of patients and, some of their operations are already implemented with DSP slices. Accordingly, since for fewer patients, the DSP utilization of the design increases, the ratio of adders in the CTUs that are implemented with DSPs is decreased so that the number of DSPs does not become the factor that limits the amount of CTUs that can be implemented.

To implement the adders in a DSP slice, the Xilinx Adder/Subtracter LogiCORE IP [63] is used. This IP is capable of implementing an adder or a subtractor (either as LUTs or with DSP48 slice) in a structure that can be pipelined with a user-defined number of stages. As such, to keep the same timing constraints, when the adders are implemented in a DSP, they are pipelined with a latency of one clock cycle, and when are implemented as logic a register is placed after the adder to introduce the same clock cycle latency. This allows that the values that are calculated in both cases to maintain the same latency and avoid the introduction of costly synchronization mechanisms.

Contingency Table Configuration

Each contingency table entry has (for a balanced dataset), a maximum value equal to half of the number of patients. Consequently, the necessary number of bits to represent each contingency table entry is given by $\lceil log_2(\#Patients/2) \rceil$. In the proposed architectures, the registers that hold the contingency table entries are implemented using only the necessary number of bits to represent their maximum value. This also impacts the number of bits that from the result values of the adders in the CTUs, from the width of the ports that are used to trasfer the contingency tables to the reconstruction units and the size of the registers and adders in the reconstruction units. By modifying the width of the contingency table entries according to the number of patients, those values are always stored, transferred and processed in the most efficient representation. Additionally, it also guarantees that an overflow does not occur, avoiding the introduction of detection and exception handling logic.

4.3.2 Mutual Information Calculation

The following Section details the implementation of the floating-point operators used for the calculation of the mutual information value for each contingency table. In particular, the operations that are performed using single-precision floating-point notation are: (i) the calculation of $n * log_2(n)$, where the input is received in an unsigned integer notation; (ii) the addiction and subtraction in the MIUs; (iii) the adder implemented in a tree structure responsible for the sum of the values outputted from the MIUs;

and (iv) the accumulator existent in the save value units.

Look-up tables

The $n * log_2(n)$ operation that is performed in the mutual information is done using a look-up table (as explained in Section 3.3.1), instead of implementing the operation itself (which would be a costly option, since the logarithm is a difficult function to implement in hardware). The look-up tables are implemented using dual-port BRAMs that are configured using a memory initialization file containing the results of the operation in a single-precision floating-point representation, for as many values as the range of the inputs. The memory initialization file is pre-generated using a C program to generate the memory initialization files. Since the look-up table that calculates the value of $n * log_2(n)$ only as one input in each MIU,it can be shared between two MIUs (by being implemented in a dual-port BRAM, instead of a single-port BRAM).

Floating-Point Operations

As mentioned above, the addiction and subtraction operations in the MIUs are preformed in the single-precision floating-point notation, as such, both operations are implemented using Xilinx floating-point LogiCORE IP [64], this IP allows for the implementation of several floating-point operations, some of which, including the addiction, subtraction and accumulation, can be implemented in a DSP48 slice and pipelined with a user-defined or automatically selected number of stages. To ensure that the timing requirements are met, the addiction and subtraction performed in the MIUs are implemented in the DSP48 slices available in the adopted Xilinx boards and configured with 11 pipeline stages.

As explained in Section 3.3.2, the number of MIUs that generate part of the same contingency table at the same time, is equal to three times the number of reconstruction units in a reconstruction block. Moreover, the resulting values (in the same clock cycle) of those CTUs must be added. This operation is performed by using several Xilinx floating-point LogiCORE IPs implemented as adders and organized in a tree structure. The resulting value is sent to the save unit where the accumulator is also deployed using the same IP core, and implemented using one DSP48 slice, with a latency of 10 clock cycles.

4.4 Accelerator Implementations

According to the previously discussed implementation details, several accelerator configurations for different numbers of patients and different orders of interactions were considered, by defining the number of reconstruction units and MIUs that are implemented for each case, depending on the number of implemented CTUs.

As a result of the design decisions that were presented in the previous Sections, the maximum amount of CTUs that can be implemented, the maximum operating frequency and the utilized bandwidth, are the parameters that define the number of combinations that can be tested in a unit of time. These parameters mainly depend on the size of the FPGA in which the design is being implemented and the

order of the combinations being tested. They are also impacted by the number of patients in the dataset, which dictates the number of reconstruction units and MIUs that can be shared for each CTU.

When generating designs targeting any order of interactions (either by using the optimizations specific for second-order interactions, or when following the general architecture for any order of interactions), the number of reconstruction units and MIUs that are implemented for each CTU is lower for higher numbers of patients (as explained in Section 3.2.4 and 3.3.2). Besides the number of reconstruction units and MIUs that are shared by CTUs, their maximum number is also influenced by the number of bits needed to represent the entries of the contingency tables, which depends on the number of patients in the data set. As such, the design performs worst when half of the number of patients is in the lower range of $[2^{(m-1)}, 2^m - 1]$, where m is the number of bits needed to represent half of the number of patients, and achieves higher performance when it is in the upper range of the interval. This happens because, within that range, the entries of the contingency tables are represented using the same amount of bits. However, as the number of patients increases (within the range), the less reconstruction units and MIUs are implemented. As such, in some particular cases, a design made for a higher number of patients (in the lower end of the $[2^{(m-1)}, 2^m - 1]$ range), may use more FPGA resources for each CTU than a design made for a lower number of patients that uses less bits to represent the table entries.

4.4.1 Second-Order Epistasis Detection Architecture

When implementing a design for second-order epistasis the optimizations that are specific to this particular case are used.

As detailed in Sections 3.2.4 and 3.3.2, the scenario where less CTUs are deployed due to the implementation of more reconstruction units and MIUs is when the number of words needed to stream an entire SNP less than 18. In that situation, which occurs from any number of patients between 1 and 512 (as $R = 64$), one reconstruction unit and one MIU are implemented for each CTU. Note that as following the specific optimizations for second-order interactions, the minimum number of words that are streamed through the AXI4 interface for each SNP is 10, which is equivalent to 320 patients. If the dataset contains less than 320 patients, dummy patients (that do not change the final result) are included. As such, from the point of view of the FPGA, 320 is the minimum number of patients supported. In this range of patients, in the ZYNQ-7 Mini-ITX, both adders present in the CTUs are implemented in a DSP slice.

For a number of words per SNP between 18 and 26 (corresponding to a number of patients between 513 and 832), one reconstruction unit and one MIU can be shared between 2 CTUs. Therefore, the maximum number of CTUs that are implemented for that range of patients increases when compared with an architecture targeting lower number of patients. This is due to the decrease in the number of reconstruction units and MIUs implemented for each CTU. For each increment of 9 SNP words, corresponding to 320 patients, one reconstruction unit and one MIU can be shared among and additional CTU.

Table 4.4, shows the number of CTUs that can share one reconstruction unit and one MIU according

to the number of patients in the dataset. The total number of reconstruction units and MIUs (when following the specific optimizations for second-order interactions) is equal, and it is calculated by dividing the number of implemented CTUs by their respective value in the Shared Units column from Table 4.4.

Table 4.4: Resource sharing for $K = 2$

Range of Patients	Range of Words per Patient	Shared Units
2 - 512	10 - 16	1
513 - 832	18 - 26	2
833 - 1088	28 - 34	3
1985 - 2240	64 - 70	7
3969 - 4288	126 - 134	14
4928 - 5120	154 - 160	17
7808 - 8000	244 - 250	27

By analyzing Table 4.4, it is possible to verify that as the number of patients increases, the less reconstruction units and MIUs are implemented. As a result, their impact in the total FPGA resource utilization decreases as the number of patients increases.

4.4.2 Third-Order Epistasis Detection Accelerator

Similarly to the specific implementation for second-order epistasis detection for the implementation of third-order epistasis detection accelerators (and for the reasons detailed in Section 3.2.4), datasets with a lower number of patients result in the implementation of less CTUs. As the number of patients in the dataset increases, the number of required reconstruction units and MIUs shared between CTUs decreases, which allows for the implementation of more CTUs.

An architecture targeting a dataset with a number of patients between 1 and 64 (which translates to 2 SNP words, since $R = 64$), results in the least number of implemented CTUs. The generated architecture has one reconstruction block, with 9 reconstruction units, shared between two CTUs, resulting in 27 MIUs and 9 reconstruction units implemented for every two CTU.

On the other hand, in an architecture targeting a dataset with 128 patients, 4 words are streamed for each SNP. Consequently, (as explained in Section 3.2.4), three clock cycles are used to stream the partial contingency tables from the CTUs to the reconstruction units. As such, each CTU is connected to one reconstruction block, each one comprising 3 reconstruction units. As the number of MIUs, is the triple of the reconstruction units (Section 3.3.2), 9 MIUs are implemented for each CTU.

The same method is adopted for every other number of patients. Table 4.5 details the number of reconstruction units and MIUs that are implemented for each CTU depending on the range of the number of patients for third-order interactions, the column labeled "*Cycles to send Table*" refers to the number of clock cycles to transport one contingency table from a CTU to the respective reconstruction unit.

The total number of reconstruction units and MIUs in a design is obtained by dividing the number of CTUs in the design by the corresponding value in the "*Contingency Units per Rec Block*" column from

Table 4.5: Resource sharing for $K = 3$

Range of Patients	Range of Words per Patient	Cycles to send Table	Contingency Units per Rec Block	Rec. Units per Rec. Block	M.I. Units per Rec. Block
2 - 64	2	1	2	9	27
65 - 128	4	3	1	3	9
129 - 256	6 - 8	3	2	3	9
257 - 512	10 - 16	9	1	1	3
513 - 832	18 - 26	9	2	1	3
833 - 1088	28 - 34	9	3	1	3
1985 - 2240	64 - 70	9	7	1	3
3969 - 4288	126 - 134	9	14	1	3
4928 - 5120	154 - 160	9	17	1	3
7808 - 8000	244 - 250	9	27	1	3

Table 4.5, which results in the total number of reconstruction unit blocks, and multiplying the result by the "*Rec Units per Rec Block*" or by "*M.I Units per Rec block*", to get the total number of reconstruction units or MIUs in the design, respectively.

4.4.3 Forth-Order Epistasis Detection Accelerator

Due to the bigger contingency tables (with 81 entries for cases and 81 for controls), that are created for fourth-order epistasis detection, the number of cycles needed to send the contingency tables from the CTU to its respective reconstruction unit is naturally higher when compared to the third-order implementation targeting the same number of patients. Therefore, the number of reconstruction units implemented for the same number of CTUs is also higher when compared to the third-order architectures, as the reconstruction units are shared among less CTUs for the same number of patients. Consequently, the number of MIUs also increases as three MIUs must be implemented for each reconstruction unit.

Table 4.6 details the number of clock cycles that are needed to stream one contingency table from the CTUs to the respective reconstruction unit, the number of contingency tables that share one reconstruction block, and the number of reconstruction units and MIUs implemented in each reconstruction block, for the different architectures that are generated depending on the number of patients in the dataset. The total number of reconstruction blocks is obtained by dividing the value in the column "*Contingency Units per Rec Block*" by the number of contingency units implemented in a design (similarly to the third-order architectures). The total number of reconstruction and MIUs is given by multiplying the values in the "*Rec Units per Rec Bloc*" and "*M.I. Units per Rec Block*" column by the total number of reconstruction blocks, respectively.

By inspection of the table, it is clear that the number of reconstruction units and MIUs decreases when the number of patients increases (as it does for all considered orders of interactions) thus improving the performance of the architecture, as more CTU can be implemented. This scenario occurs unit the number of patients allows for the implementation of only one reconstruction block shared between all CTUs in the design. As is the case for any accelerator generated using the proposed architecture, the

Table 4.6: Resource sharing for $K = 4$

Range of Patients	Range of Words per Patient	Cycles to send Table	Contingency Units per Rec Block	Rec Units per Rec Block	M.I. Units per Rec Block
2 - 64	2	1	2	27	81
65 - 128	4	3	1	9	27
129 - 256	6 - 8	3	2	9	27
257 - 512	10 - 16	9	1	3	9
513 - 832	18 - 26	9	2	3	9
833 - 1664	28 - 52	27	1	1	3
1665 - 2560	54 - 80	27	2	1	3
2561 - 3392	82 - 106	27	3	1	3
2561 - 4288	108 - 134	27	4	1	3

number of contingency tables grows with the number of patients in the targeted dataset in a logarithmic fashion, as the impact in for the total amount of used FPGA resources of the number of implemented reconstruction units and MIUs decreases as the number of patients increases.

4.5 Summary

This chapter detailed regarding the design choices made for the implementation of accelerators based on the proposed architecture, and regarding the platform used for their implementation. In the first section details were provided regarding the general organization of the targeted Zynq SoC platform, including details regarding the available FPGA resources, along with some other relevant characteristics of the adopted boards. The section also provided details on alternative FPGA boards targeting high-performance systems with physically separated Host CPU and FPGA accelerator.

Next, design choices regarding the implementation of some of the functional blocks used in the proposed architecture were discussed, both and in the creation of the contingency tables in the calculation of the mutual information. Finally, several details were discussed regarding the chosen parameters and the number of shared resources for the implementation of accelerators for second, third, and fourth-order epistasis detection targeting datasets with different numbers of patients.

Chapter 5

Experimental Results

In order to fully understand the capabilities of the proposed accelerator architecture for high-order epistasis detection, in this Chapter it is performed a thorough performance and energy efficiency evaluation and comparison against the other state-of-the-art FPGA-based accelerators [21, 22, 56, 57]. Accordingly, experimental results for resource utilization, execution times, and estimated power and energy consumption are discussed for several accelerator implementations.

For the performed evaluation, besides the execution time and the energy consumption, a specific metric topically used in epistasis detection studies [16], was relied upon in order to evaluate the performance of the implemented architectures and compare them against the state-of-the-art. In particular, it was considered the number of unique combinations that are processed per second, normalized to the number of patients, calculated as:

$$\#Tests * \frac{\#Patients}{time(s)}, \tag{5.1}$$

where $\#Tests$ represents the total number of unique combinations to be tested. The adopted normalization to the number of patients ensures that the obtained results can be compared against other implementations using datasets with a different numbers of patients.

Additionally, to measure the energy efficiency of the proposed architectures and compare it against the state-of-the-art implementations a performance-energy efficiency metric was adopted, in the from of an Energy Delay Product (EDP), calculated by multiplying the energy consumption with the execution time.

5.1 Implementation and Performance Results

This section details the implementation results and evaluates the performance of different second, third, and fourth-order accelerator configurations devised from the proposed architecture, targeting different numbers of patients. The second and third-order accelerators are implemented in configurations targeting numbers of patients ranging from 64 to 8000, and several datasets of up to 500 000 and 20 000 SNPs, for the second and third-order accelerators, respectively, are used to evaluate their perfor-

mance. The fourth-order accelerators are implemented for numbers of patients ranging from 64 to 4000 and evaluated with datasets of up to 2000 SNPs, due to the much higher execution time. The mentioned accelerators are implemented in a ZYNQ7-Mini ITX board (Zynq-7000 SoC), and some configurations are compared against similar accelerators implemented in a Zynq-Ultrascale+ ZCU102 board (Zynq-Ultrascale+ SoC) and in a Virtex-7 690T FPGA.

All the referred implementations are made using Xilinx Vivado 2017.2 design suite, the obtained execution times were measured in the board and the power and energy estimations were obtained though the Vivado power estimator.

5.1.1 Second-Order Accelerator Analysis

This section details the implementation results of several second-order accelerator configurations (targeting a number of patients ranging from 500 to 8000) in a ZYNQ7-Mini ITX board containing a Zynq-7000 SoC and evaluates their performance for different datasets. The obtained results are compared against accelerator configurations for 4000 and 5000 patients implemented in a Zynq-Ultrascale+ SoC and a Virtex-7 9690T FPGA.

Hardware Resources: ZYNQ7 Mini-ITX

As mentioned before, in the proposed architecture, the number of CTUs in the design increases with the number of patients. Consequently, the number of contingency tables that are processed in a unit of time (when normalized to the number of patients), also increases, up until the point where only one reconstruction and MIU are implemented in the entire design. When that happens, the number of CTUs remains constant and so does that metric.

Accordingly, Table 5.1 details the maximum number of CTUs that can be implemented in the ZYNQ-7 Mini-ITX board, for datasets with different numbers of patients, as well as the number of FPGA resources that are used to implement each configuration. Since, the amount of CTUs that are implemented is considerably high, and as the number of BRAMs that is used is proportional to the number of units and to the number of patients, the number BRAMs existent on the FPGA is easily exceeded, as such, the buffers that hold the data referring to one SNP in each unit, are implemented using logic, impacting the total number of implemented CTUs (when the number of BRAMs in the board is fully occupied).

Table 5.1: Implementation results in a Zynq-7000 SoC of $K = 2$ architectures

Number of Patients	Units	Power (W)	LUT (%)	DSP (%)	BRAM (%)
500	136	5.46	210224 (76)	1632 (80)	204 (27)
1000	270	10.23	199420 (72)	1800 (89)	675 (89)
2000	392	11.04	212472 (77)	2016 (100)	755 (100)
4000	420	8.31	214567 (77)	1920 (95)	755 (100)
5000	391	9.14	221749 (80)	1748 (87)	755 (100)
8000	422	9.24	229278 (82)	1816 (90)	755 (100)

This is evidenced by the results obtained, where it is possible to verify that the variation of the

maximum number of CTUs with the number of patients is less pronounced as the number of patients increases (see Table 5.1). However, when comparing architecture configurations for 4000 and 5000 patients, the maximum number of contingency tables decreases. This happens due to an increase in the number of bits needed to represent the entries of the contingency tables. In turn, this leads to an increase in the LUT utilization of both the CTUs and the reconstruction units, without a substantial decrease in the number of reconstruction and MIUs that are implemented (as explained in Section 4.4).

By observing the obtained results, is evident that the power consumption of the accelerator is dependent on the FPGA resource utilization. The accelerator that uses less power (for 500 patients) is implemented using only 27% of the available BRAMs, while the accelerator that uses more power (for 2000 patients) fully utilizes the available DSPs and BRAMs of the FPGA.

Performance Analysis: ZYNQ7 Mini-ITX

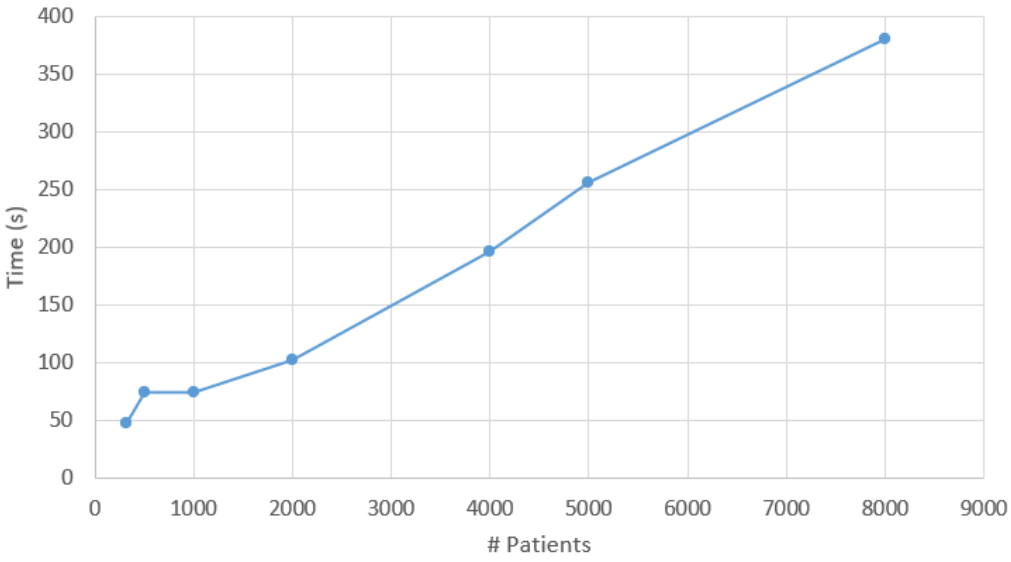

Figure 5.1: Execution time of $K = 2$ epistasis when changing the number of patients

The graphs presented in Figures 5.2, 5.3, and 5.1 represent the evolution of the elapsed time to perform second-order epistasis detection with the proposed architecture. The implemented architecture makes use of the specific optimizations for second-order interactions and targets a balanced dataset with 5000 patients. Figure 5.2 showcases the inversely proportional relation existent between the number of implemented CTUs and the execution time and EDP, using a dataset with 500 000 SNPs. The latter follows a similar pattern to the execution time. This happens because, although an implementation with less CTUs draws less power, the power used by the PS is constant. As such, the designs with less CTUs have a higher total energy consumption, due to their higher execution time.

Figure 5.3 showcases the execution time as the number of SNPs in the used dataset increases from 10 000 to 500 000, with 391 implemented CTUs. The observed growth of the execution time happens because the number of combinations to be processed (given by Equation 2.1), grows, for second-order interactions, with the square of the number of SNPs. Therefore, as the number of SNPs increases, the

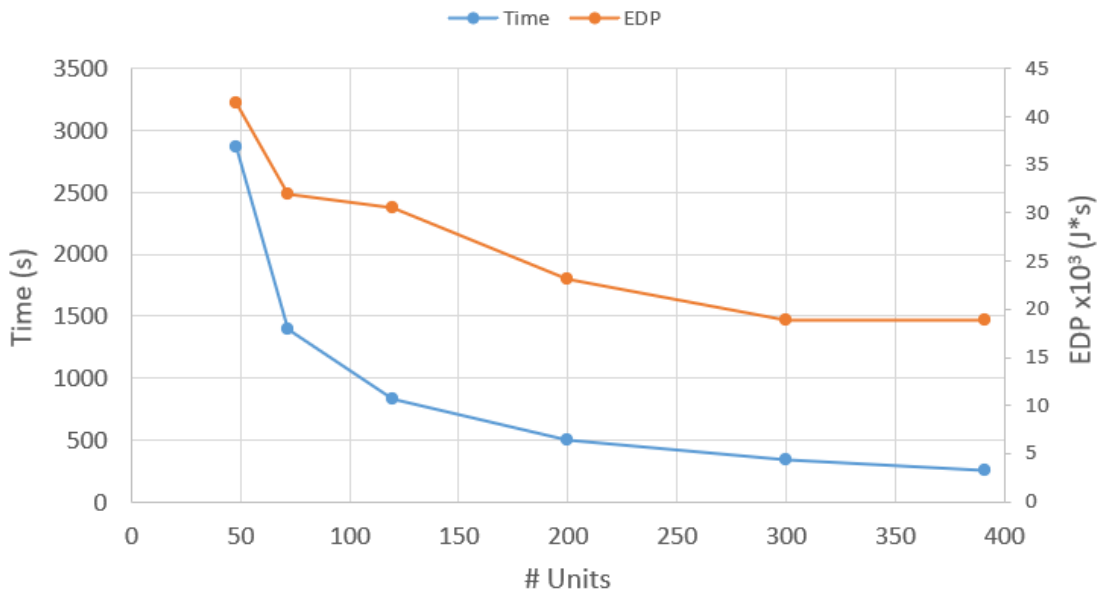

Figure 5.2: Execution time and EDP of $K = 2$ epistasis when changing the number of units

number of rounds to be tested increases by a power of two and, consequently, so does the execution time.

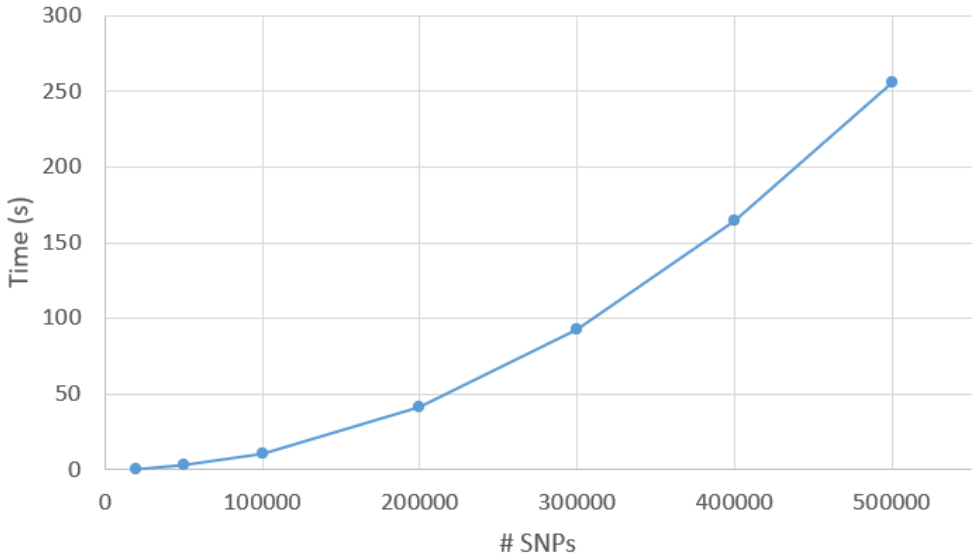

Figure 5.3: Execution time of $K = 2$ epistasis when changing the number of SNPs

Figure 5.1 represents the growth of the execution time when performing second-order epistasis detection on a dataset with the same number of SNPs, 500 000 in this case, but for a different numbers of patients. As the number of patients does not impact the number of combinations that are processed, only the required number of words to stream one SNP from the board memory to the FPGA through the AXI4 interface, the increase in the execution time with the number of patients is linear. However, as illustrated in Figure 5.1, despite having a linear behavior, the slope of the graph suffers slight variations due to the higher number of CTUs that can be implemented for a higher number of patients.

Therefore the slope between the first two points of the graph, corresponding to 320 and 500 patients, where both architectures contain the same number of CTUs, is steeper than in the rest of the graph, as between the other points (except between 4000 and 5000 patients), the number of CTUs that can be implemented increases with the number of patients.

Comparison with Zynq-Ultrascale+ ZCU102

Table 5.2 represents the results of performing second-order epistasis detection on a balanced dataset containing 4000 patients and 500 000 SNPs using the Zynq-7000 and the Zynq-UltraScale+ SoCs. When using a Zynq-UltraScale+ SoC, a total of 434 CTUs, running at 322MHz, can be implemented. This results from the fact that, despite having slightly less available LUTs than the Zynq-7000 (which can only implement 420 of those units), this board contains more BRAMs. Therefore, for second-order implementations, fewer buffers to hold the SNP data in each unit are implemented using logic, which results in the implementation of slightly more CTUs. However, the higher clock frequency supported by the Zynq UltraScale+ ZCU102 board is what plays the most important part in the $1,34\times$ higher throughput that is archived when compared to the ZYNQ-7 Mini-ITX. The design implemented in the Zynq UltraScale+ ZCU102 board has a higher power consumption than the design implemented in the ZYNQ-7 Mini-ITX, in the most part because of the higher clock frequency at which the design runs, but also due to the higher number of implemented CTUs. Nevertheless, as the execution takes less time, the total energy consumption is similar in both boards. Naturally, the more efficient FPGA fabric in the Zynq-Ultrascale+ SoC provides greater energy efficiency than the 28nm technology of the Zynq-7000 SoC fabric, which is verifiable by the lower EDP achieved by the first SoC while implementing similar designs. This relation between the FPGA fabric and the EDP is observable in all implementations of similar designs in the Zynq-7000 and Zynq-Ultrascale+ SoCs.

Table 5.2: Results of $K = 2$ architecture implemented in different boards for 4000 patients

Device	Units	Frequency (MHz)	Time (s)	Comb/sec x10^{12}	Power (W)	Energy (kJ)	EDP x10^{6} (J*s)
Zynq 7000	420	250	3min 11s	2.62	8.31	1.58	0.30
Zynq Ultrascale+	434	322	2min 23s	3.50	12.62	1.80	0.26

Table 5.3 provides similar results as Table 5.2 for a dataset with the same number of SNPs (500 000) but with 5000 patients. As expected, the performance of the architecture targeting 5000 patients, measured by the number of combinations processed per second normalized to the number of patients, is inferior to the performance obtained when targeting 4000 patients, due to the fewer number of implemented CTUs, resulting in the increase in the number of bits needed to represent one entry of the contingency tables.

Table 5.3: Results of $K = 2$ architecture implemented in different boards for 5000 patients

Device	Units	Frequency (MHz)	Time (s)	Comb/sec x10^{12}	Power (W)	Energy (kJ)	EDP x10^{6} (J*s)
Zynq 7000	391	250	4min 16s	2.44	9.14	2.34	0.60
Zynq Ultrascale+	408	322	3min 10s	3.29	11.25	2.12	0.40
Virtex-7 690T	612	250	2min 44s	3.98	7.69*	1.26*	0.21*

Comparison with Virtex-7 609T

The same design, targeting a dataset with 5000 patients, was also implemented in a Virtex-7 690T FPGA, for better comparison against the state-of-the art-implementation that used the same FPGA [56]. As the Virtex-7 690T is a bigger FPGA than the ones contained in both of the used Zynq SoCs, a higher amount of CTUs can be implemented. The increase in the implemented CTUs results in a speedup of 1.63$\times$ and 1.16$\times$ when compared to the Zynq-7000 and Zynq-Ultrascale+ SoCs, respectively. The design implemented in the Virtex-7 690T runs at the same 250MHz clock speed as the Zynq-7000 as they use the same 28mn technology for the FPGA fabric.

Despite having more CTUs, the design implemented in the Virtex-7 FPGA shows a lower power consumption than the design implemented in the Zynq SoCs. This happens because the power that is reported by Vivado for the Virtex-7 implementation only takes into consideration the design implemented in the FPGA, whereas the power reported for the Zynq SoCs also accounts for the power consumption of the PS, which, in the Zynq-Ultrascale+, requires more than 3W of power. Therefore, the power and energy comparisons between the complete system implemented in a Zynq SoC and a standalone design implemented on an FPGA (marked with an asterisk on the tables) is not conclusive, as the latter has to be connected to a host computer, to manage the data transfers to the FPGA, which would draw a significant amount of power. Nonetheless, the obtained results are still important references for comparison with the state-of-the-art solutions.

5.1.2 Third-Order Architecture

This section details the implementation results and the evaluation of the performance of different third-order accelerator configurations (for different numbers of patients, from 64 to 8000) in a ZYNQ7-Mini ITX board (containing a Zynq-7000 SoC), and compares the results for 4000 and 5000 patients against a similar accelerator implemented in a Zynq-Ultrascale+ SoC and a Virtex-7 690T FPGA.

Hardware Resources: ZYNQ7 Mini-ITX

Table 5.4 shows the number of CTUs implemented in the ZYNQ-7 Mini-ITX broad for different numbers of patients, as well as the power and the FPGA resources required by each of the implemented designs, which run, in this FPGA, at 250MHz. Similarly to the second-order implementations (and for the reasons explained in Sections 4.4 and 4.4.1), the increase of the maximum number of CTUs with the number of patients is visible. However, it is less pronounced as the number of patients increase, and

ceases to exist when only one reconstruction unit and three MIUs are shared among all CTUs.

Table 5.4: Implementation results in a Zynq-7000 SoC of $K = 3$ architectures

Number of Patients	Units	Power (W)	LUT (%)	DSP (%)	BRAMS (%)
64	20	17.44	195117 (70)	1880 (93)	410 (54)
250	46	17.54	236136 (85)	1886 (93)	322 (43)
500	57	16.27	230115 (83)	1767 (87)	285 (38)
800	80	14.93	226948 (82)	1680 (83)	200 (26)
2000	126	14.65	238381 (86)	1782 (88)	380 (50)
4000	140	15.21	250894 (90)	1900 (94)	422 (56)
5000	130	14.28	233983 (84)	1866 (92)	438 (58)
8000	140	15.05	253903 (92)	1952 (97)	444 (59)

Performance Analysis: ZYNQ7 Mini-ITX

Figures 5.4 and 5.5 showcases the evolution of the execution time when varying the number of SNPs and the number of implemented CTUs, respectively, when using an architecture targeting third-order interactions. The architecture is implemented in the ZYNQ7 Mini-ITX board featuring a Zynq-7000 SoC, for a balanced dataset containing 5000 patients.

Similarly to the second-order architectures, the execution time grows with the number of SNPs in the dataset. However, instead of the number of combinations to be tested growing with the square of the number of SNPs, for third-order interactions it grows with the cube (as reflected by Equation 2.1, in Chapter 2). Therefore, the higher the order of interactions, the bigger is the variation in the number of combinations to be tested, inducing a grater penalty in the execution time. The inversely proportional relation between the number of units and the execution time, as well as the EDP measurement for the second-order architecture, also occurs for the third-order architecture, as visible in Figure 5.5.

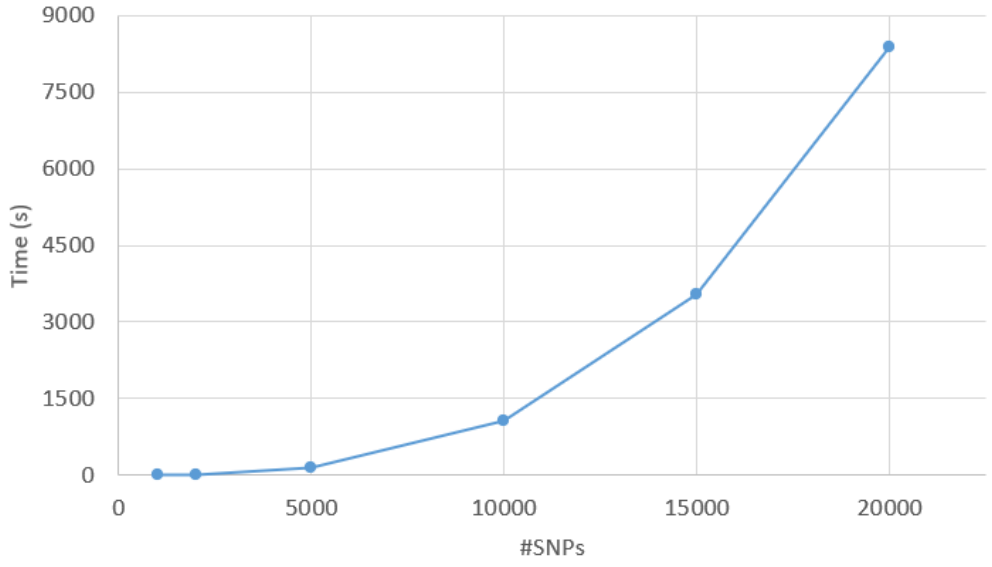

Figure 5.4: Execution time and EDP of $K = 3$ epistasis when changing the number of SNPs

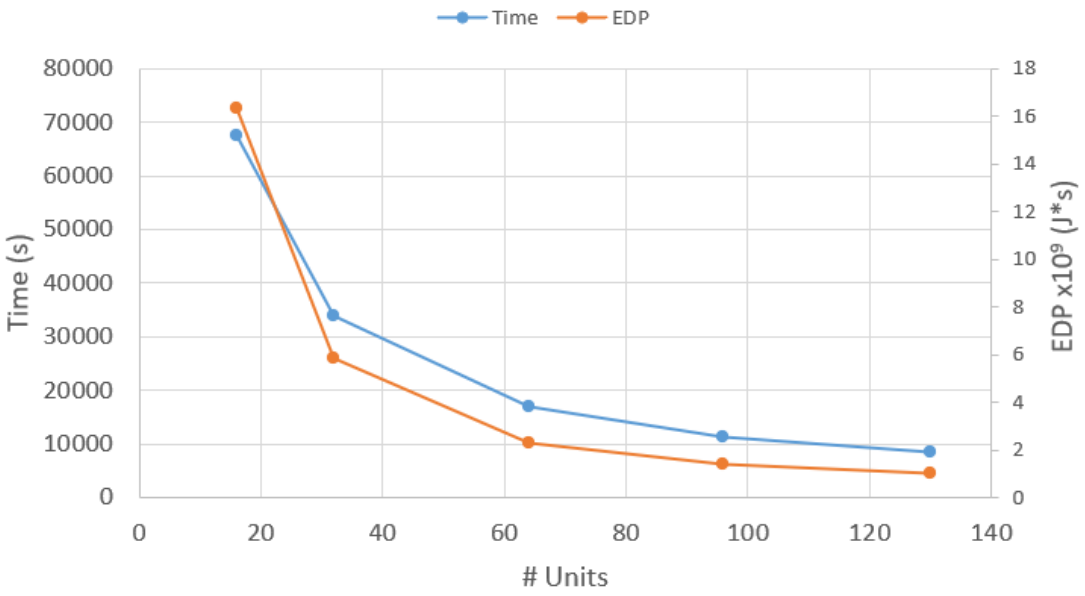

Figure 5.5: Execution time of $K = 3$ epistasis when changing the number of units

The linear growth of the execution time with the number of patients observable for the second-order architecture (explained in Section 5.1.1), is, as depicted by Figure 5.6, also valid for the third-order architectures. Similarly to the second-order implementations, despite the amount of data to transfer varying linearly with the number of patients, their relationship with the execution time is not perfectly linear, as the number of contingency tables that are implemented change depending on the number of patients in the dataset that it targets. Therefore, the slope of the graph changes depending on the variation of the number of CTUs that are implemented.

For example, similarly to the second-order implementations (shown in Section 5.1.1), the number of CTUs implemented in an architecture generated to target a dataset with 4000 patients is higher than one that targets a dataset with 5000 patients, consequently, the slope between those two points is steeper than between, for example, 5000 and 8000 patients, as the number of CTUs that is implemented in an architecture targeting 8000 patients is higher than in one targeting 5000.

Comparison with Zynq-Ultrascale+ and Virtex-7 690T

A third-order architecture targeting 4000 patients was also implemented in the Zynq-Ultrascale+ SoC as well as in one Virtex-7 690T FPGA as for the second-order accelerator, for the later, only the FPGA design was considered, contrarily to the implementations in the Zynq SoCs, where a fully functional system is created. Table 5.5 represents the maximum number of CTUs in the implemented architectures in different devices, along with the execution time and the energy consumption to preform third-order epistasis detection on a dataset with 20 000 SNPs and 4000 patients in each of the boards.

The power consumption of each design is impacted not only by the resource utilization but also by the frequency at which the design runs, consequently, the design implemented in the Zynq-Ultrascale+

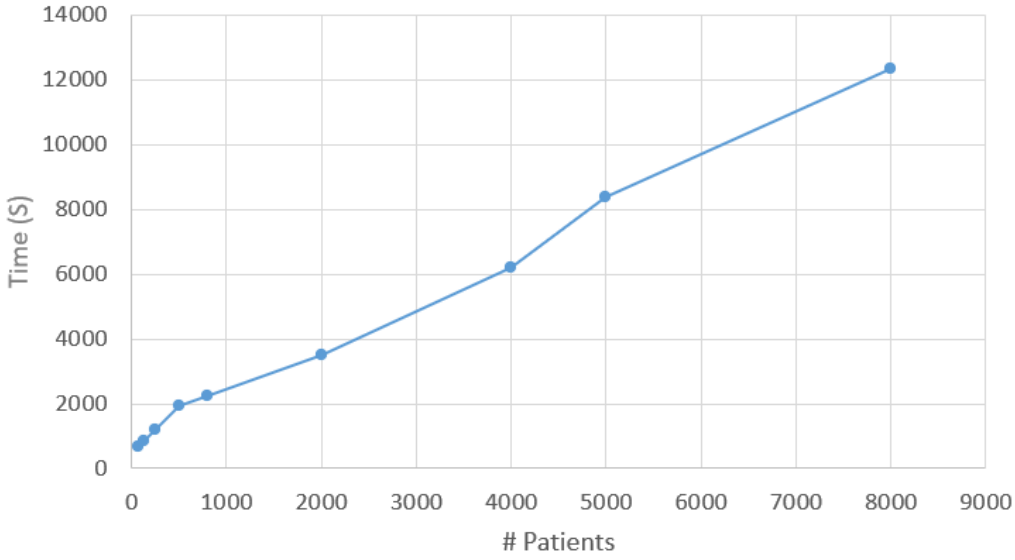

Figure 5.6: Execution time of $K = 3$ epistasis when changing the number of patients

Table 5.5: Results of $K = 3$ architecture implemented in different boards for 4000 patients

Device	Units	Frequency (MHz)	Time (s)	Comb/sec x10^{12}	Power (W)	Energy (kJ)	EDP x10^{6} (J*s)
Zynq-7000	140	250	1h 43min	0.86	15.21	94.00	580.92
Zynq-Ultrascale+	135	322	1h 23min	1.07	16.70	83.16	414.14
Virtex-7 690T	220	250	1h 6min	1.34	22.72*	90.43*	358.10*

ZCU102 board has a higher power consumption than the one implemented in the ZYNQ7-Mini-ITX, even though fewer CTUs are implemented. Despite having less CTUs, the architecture implemented in the Zynq-Ultrascale+ ZCU102 archives a speedup of $1.25\times$, when compared to ZYNQ7-Mini-ITX, due to its higher clock frequency. The smaller execution time also explains the lower energy consumption observed in the Zynq-Ultrascale+ when compared to the Zynq-7000 SoC, even though the design implemented on the latter requires less power. The smaller transistor technology (16nm vs 28mn), also plays a part in the smaller energy consumption of the Zynq-Ultrascale+ ZCU102.

The Virtex-7 690T, being an all-round larger FPGA, allows for the implementation of more contingency tables. Consequently, the design implemented in this FPGA requires more power than the one implemented in the Zynq-7000 SoC, as both run at the same clock frequency, and even than the one implemented in the Zynq-Ultrascale+ SoC due to its bigger size. The design implemented in the Virtex-7 690T FPGA provided a $1.56\times$ and $1.25\times$ higher throughput when compared to the architectures implemented in the Zynq-7000 and the Zynq-Ultrascale+ SoCs, respectively. The power and energy values obtained for the Virtex-7 FPGA are only taking into account the power consumption of the FPGA design, whereas the values obtained for the Zynq-7000 and Zynq-Ultrascale+ based boards also take into account the power consumption of the Processing System integrated into the SoC.

Table 5.6 presents the results of performing third-order epistasis detection over a dataset containing, 20 000 SNPs and 5000 patients. Comparing to the architecture targeting a dataset with 5000 patients

(Table 5.5), there is, as expected, a slight performance drop, as the number of CTUs that are implemented decreases.

Table 5.6: Results of $K = 3$ architecture implemented in different boards for 5000 patients

Device	Units	Frequency (MHz)	Time (s)	Comb/sec x10^{12}	Power (W)	Energy (kJ)	EDP x10^{6} (J*s)
Zynq-7000	130	250	2h 20min	0.80	14.28	119.95	1007.58
Zynq-Ultrascale+	125	322	1h 53min	0.98	15.71	106.52	722.21
Virtex-7 690T	204	250	1h 29min	1.24	20.06*	107.10*	539.78*

5.1.3 Forth-Order Architecture

This section evaluates the implementation and performance of fourth-order accelerator configurations targeting numbers of patients ranging from 64 to 4000, implemented in a ZYNQ7-Mini ITX board (Zynq-7000 SoC), and compares the configuration targeting 4000 patients with a similar accelerator implemented in a Zynq-Ultrascale+ SoC and in a Virtex-7 690T FPGA.

Hardware Resources: ZYNQ7 Mini-ITX

Table 5.7 shows the maximum number of contingency tables implemented depending on the number of patients in the targeted dataset for the Zynq-7000 SoC running at 250 MHz, as well as the required power of the design and the FPGA resources that are needed to implement it. Similarly to the implementations for lower-order epistasis detection, the number of implemented CTUs grows with the increase in the number of patients.

Table 5.7: Implementation results in a Zynq-7000 SoC of $K = 4$ architectures

Number of Patients	Units	Power (W)	LUT (%)	DSP (%)	BRAM (%)
64	6	15.50	175635 (63)	1674 (83)	366 (49)
128	10	18.40	223334 (81)	1950 (97)	410 (54)
320	20	15.41	232422 (84)	1940 (96)	280 (37)
640	26	13.88	232127 (84)	1742 (86)	182 (24)
1000	34	12.70	235541 (85)	1870 (93)	242 (32)
2000	40	12.75	244217 (88)	1840 (91)	224 (30)
4000	48	13.72	210875 (76)	1704 (84)	268 (36)

Performance Analysis: ZYNQ7 Mini-ITX

Similarly to the results obtained in lower-order epistasis detection (presented in Section 5.1.1 and Section 5.1.2, for second and third-order, respectively), the execution time of an accelerator generated using the proposed architecture targeting fourth-order interactions (as shown in Figure 5.7), or for any other order of interactions, the relation between the execution time (represented in minutes) and the EDP, and the number of implemented CTUs is inversely proportional. Figure 5.8 showcases the growth

of the execution time with the number of SNPs in the dataset, as it happens for every order of interactions. However, for fourth-order epistasis detection, the number of unique combinations to be tested grows with the fourth of the number of SNPs, as such, the execution time grows at the same rate (as defined in Equation 2.1.

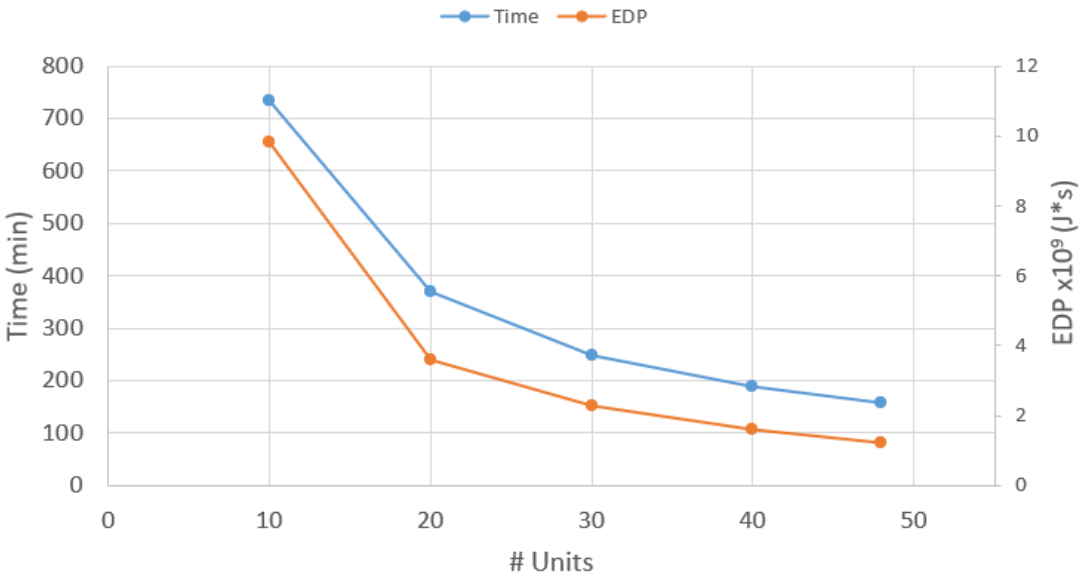

Figure 5.7: Execution time and EDP of $K = 4$ epistasis when changing the number of units

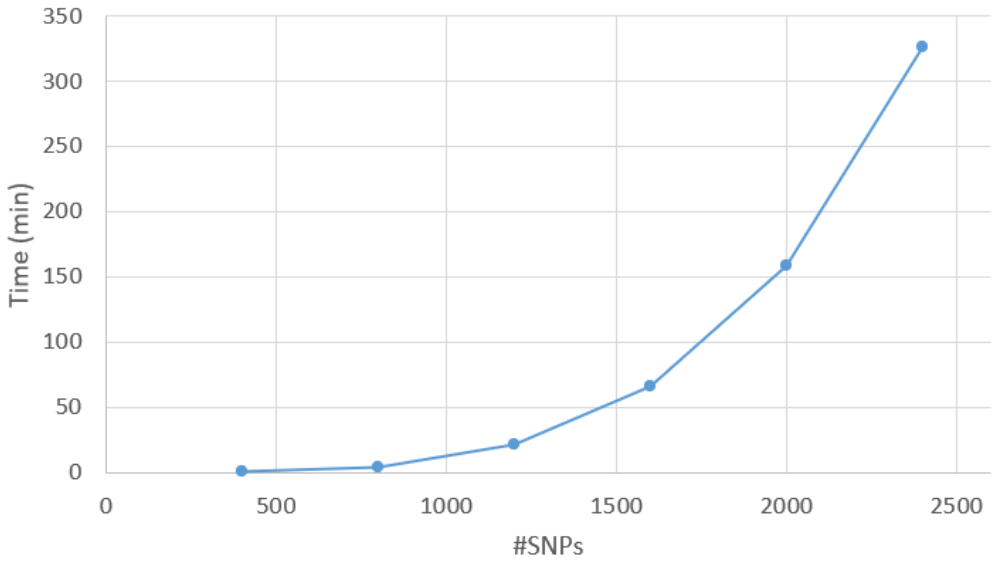

Figure 5.8: Execution time of $K = 4$ epistasis when changing the number of SNPs

Figure 5.9, shows the increase of the execution time (in minutes) when performing forth-order epistasis detection on a dataset with 2000 SNPs, using a number of patients ranging from 64 to 4000. As the difference in the number of patients contained in the dataset does not change the number of unique combinations, the change in the execution time with the number of patients should be linear for the

same amount of CTUs. However (as is the case for all orders of interactions), the maximum number of implemented CTUs, increases with the number of patients, which explains the increasingly less steep slope of the graph, instead of the expected linear behavior.

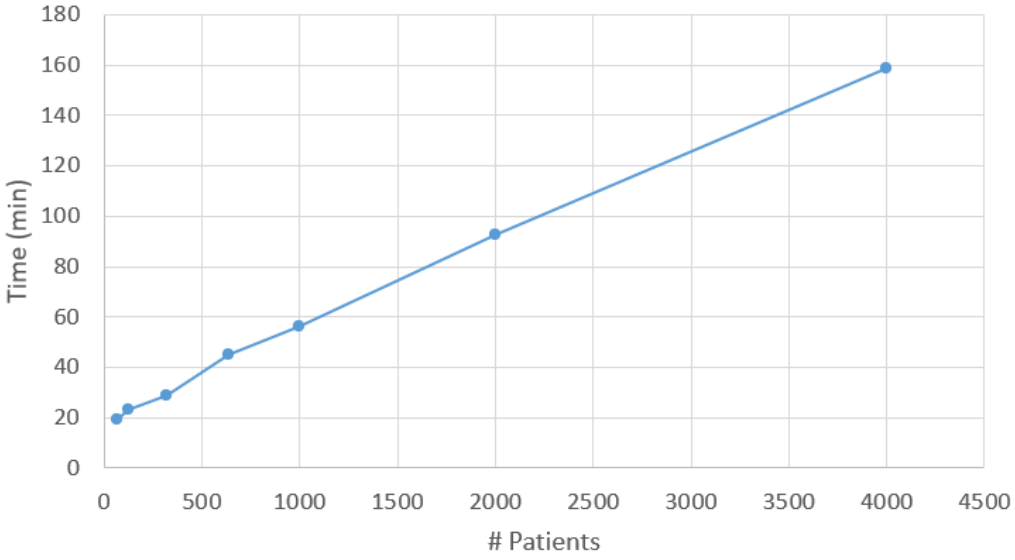

Figure 5.9: Execution time of $K = 4$ epistasis when changing the number of patients

Comparison with Zynq-Ultrascale+ and Virtex-7 690T

Table 5.8 details the results of performing forth-order epistasis detection over a dataset containing 4000 patients and 2000 SNPs using the considered FPGA boards. The design implemented in the ZYNQ7 Mini-ITX board is the same that is represented in Table 5.7, runs at 250MHz and contains 48 CTUs. On the other hand, the design implemented in the Zynq-Ultrascale+ ZCU102 runs at 322MHz, as the board contains a more recent FPGA fabric, which allows for higher clock speeds. However, as the FPGA has slightly less available LUTs, only 46 CTUs are implemented.

Similarly to the implementations done for third-order interactions (described in Section 5.1.2), despite containing fewer CTUs, due to its higher clock frequency, the design implemented in the Zynq-Ultrascale+ ZCU102 for fourth-order epistasis detection, requires more power than the one implemented in the ZYNQ7 Mini-ITX. However, as the throughput is 1,25 time higher, also due to the higher clock frequency, the total energy consumption ends up being lower in the Zynq-Ultrascale+ ZCU102 than in the ZYNQ7 Mini-ITX when preforming forth-order epistasis using the same dataset.

The generated fourth-order architecture was also implemented in a Virtex-7 690T FPGA, as this FPGA uses the same 28nm FPGA fabric as the Zynq-7000 SoC, the design runs at the same clock frequency of 250MHz. The higher number of resources available in this FPGA allows for the implementation of more CTUs, enabling the execution of a fourth-order epistasis detection, in under two hours for a dataset with 2000 SNPs and 4000 Patients, resulting in speedups of 1.5 and $1.2\times$ when compared to the Zynq-7000 and Zynq-Ultrascale+ SoCs, respectively. Although the design implemented in the Virtex-7 uses more power than the designs implemented on the other boards, as it implements more

CTUs, the total energy consumption is lower, due to the smaller execution time when compared to the Zynq-7000 SoC. However, it consumes more energy than the Zynq-Ultrascale+ SoC, as this SoC makes uses of a more power efficient 16mn FPGA fabric.

Table 5.8: Results of $K = 4$ architecture implemented in different boards for 4000 patients

Device	Units	Frequency (MHz)	Time (s)	Comb/sec x10^{12}	Power (W)	Energy (kJ)	EDP x10^{6} (J*s)
Zynq 7000	48	250	2h 38min	0.28	13.72	130.07	1233.06
Zynq Ultrascale+	46	322	2h 8min	0.35	14.83	114.12	876.44
Virtex-7 690T	74	250	1h 46min	0.42	19.50*	123.41*	784.89*

5.2 Discussion

In conclusion, by following the proposed method, that is detailed in Chapter 3, specialized architecture to detect epistasis targeting any order of interactions and adaptable to any number of patients can be easily created, as demonstrated in Chapter 4. As explained in Section 4.1, the complete proposed system can be implemented in any board featuring a Zynq SoC. However, the generated logic designs can be implemented in any modern FPGA. The generated architectures when implemented in larger FPGAs, can be implemented with more contingency table units. This leads to (as corroborated in Section 5.1 by the implementations on the Virtex-7 690T) lower execution times and higher performance, whcih means that the proposed architecture is scalable with the amount of FPGA resources available.

Naturally the execution time for the same number of patients increases with the order of interactions targeted, as the number of combinations to be evaluated increases dramatically and the number of implemented CTUs decreases, as each CTU is more complex and the number of implemented MIUs and reconstruction units increases. Therefore, the energy efficiency of the implemented accelerators (measured by the EDP) is lower for higher-order interactions. The lower BRAM utilization of the third and fourth-order implementations when compared to the second-order implementations (where it is near 100%), is a direct consequence of the CTU in the systolic array (except the first) only storing one SNP, independently of the order of interactions. As such, the number of BRAMs that are implemented for each CTU is the same for every order of interactions (neglecting the first unit). This results in a higher BRAM utilization in the second-order architectures (as more CTUs are implemented) than for architectures targeting higher-order interactions.

5.3 Comparison with State-of-the-art Accelerators

This section provides a comparison of the obtained results when performing epistasis detection with the proposed architectures, with the results obtained by the comparable state-of-the-art implementations in FPGA [21, 22, 56, 57]. The metric used to compare the performance of the designs is the number of combinations tested per second normalized to the number of patients, along with the execution time

and the power consumption. The comparisons in this Section are restricted to second and third-order epistasis detection as, at the time of writing, no other implementations of higher-order epistasis detection based on an exhaustive search method are available to be compared against.

5.3.1 Second-Order Epistasis Accelerators

The state-of-the-art implementation for second-order epistasis detection proposed in [21] (detailed in Section 2.4.1), is implemented in the RIVYERA S6-LX150 accelerator, which contains 128 Xilinx Spartan6-LX150 FPGAs, targeting a dataset with 5000 patients, and it is used to run a dataset with 500 000 SNPs. The results referring to the state-of-the-art implementations and the results obtained by the proposed architectures implemented in a Zynq-7000 and Zynq-Ultrascale+ SoCs, for a dataset with the same size are represented in Table 5.9.

The proposed design was not implemented in the Spartan-6 FPGA, which would provide a better comparison against the state-of-the-art architecture, because the it was not available during the work performed for this thesis and it is not supported by Vivado. Therefore, comparing the performance of both architectures is not easy as they run at different clock frequencies, are implemented in FPGAs with different technologies, and rely on different objective functions. Not with standing, the gains in terms of energy consumption are clear, as the proposed architecture achieves a similar execution time using only one Zynq-7000 SoC, as opposed to the $128\times$ FPGAs used in the state-of-the-art implementation, which results in an energy consumption about $77\times$ and $84\times$ lower in the Zynq-7000 and the Zynq-Ultrascale+ SoCs, respectively. The differences in EDP of the state-of-the-art and the proposed architecture are similar to the differences of energy consumption, as the execution time is identical, which means that the proposed architecture is $77\times$ and $81\times$ times more energy efficient, when implemented in a Zynq-7000 and the Zynq-Ultrascale+ SoC, respectively.

It is worth noting that the state-of-the-art implementation uses iLOCi [12] as the objective function, while the proposed architecture uses the mutual information. However (as detailed in Section 4.4.1), for 5000 patients the resources used by the MIUs are much less relevant for the total than the resources used the CTUs, and, the iLOCi function involves similar computational complexity when compared to the mutual information. Therefore, the use of iLOCi as the objective function would not have a significant impact in the performance of the design.

Table 5.9: $K = 2$ results for a dataset containing 500 000 SNPs and 5000 patients

Device	Time	Power (W)	Energy (kJ)	Comb/sec x10^{12}	EDP x10^{6} (J*s)
RIVYERA S6-LX150[21]	4min	780	50.0	2.61	43.20
Heterogeneous system[56]	12min	250	50.0	0.87	129.60
Zynq-7000 [Proposed]	4min 16s	9.14	0.65	2.44	0.60
Zynq-Ultrascale+ [Proposed]	3min 10s	11.25	0.59	3.29	0.40
Virtex-7 690T [Proposed]	2min44s	7.69*	0.35*	3.98	0.21*

The heterogeneous system represented in Table 5.9 refers to the FPGA + GPU implementation

proposed in [56], where the entire FPGA is used to create the contingency table, as the GPU is used to calculate the objective function, which, in this case is the BOOST filter [11]. The components used to implement this solution are a Virtex-7 690T FPGA paired with a GeForce GTX 780 Ti GPU, both connected to a Host computer. The generation of the contingency tables in the FPGA is based on the same architecture proposed in [21], but implemented in a modern Virtex-7 FPGA, which allows for a better comparison against the architecture proposed in this thesis (despite the different objective function that is used).

Despite having the entire FPGA resources available to the generation of the contingency tables, the throughput of the heterogeneous system is $4.57\times$ lower than that of the architecture proposed in this thesis, when implemented in the same Virtex-7 690T FPGA. The power and energy values of the proposed implementation in a similar Viterx-7 FPGA are marked with an asterisk since they are not taking into account the power consumption of a Host computer, that would be needed to implement a complete system. As such, the energy efficiency comparisons between the heterogeneous system and the architecture proposed in this thesis is done by taking into account the values obtained in the Zynq-700 SoC, as it used the same FPGA fabric and works at the same clock frequency as the Virtex-7 690T FPGA. As indicated by the $216\times$ lower EDP, the use of a single FPGA is much more energy efficient than the combination of an FPGA and a GPU.

5.3.2 Third-Order Epistasis Accelerators

The final comparison considers the architecture for third-order epistasis detection proposed in [22] (described in Section 2.4.2). The design is implemented in two different FPGAs, a Knitex-7 325T and a Virtex-7 680T. This Section is focused on the latter as it is the one that offers the best performance due to its larger size. Similarly to the accelerator proposed in the thesis, this architecture also uses the mutual information as the objective function, which allows for a direct comparison between both.

Table 5.10 shows the results achieved by the state-of-the-art third-order FPGA accelerator and by an accelerator targeting a dataset of the same size generated with the architecture proposed in the thesis. The latter is implemented in a Zynq-7000 and in a Zynq-Ultrascale+ SoC, and in a Virtex-7 690T FPGA, for better comparison against the state-of-the-art architecture.

Table 5.10: $K = 3$ results for a dataset containing 20 000 SNPs and 5000 patients

Architecture	Time	Comb/sec x10^{12}
Virtex-7 690T[22]	4h 33min	0.41
Zynq-7000 [Proposed]	2h 20min	0.80
Zynq-Ultrascale+ [Proposed]	1h 53min	0.98
Virtex-7 690T [Proposed]	1h 29min	1.24

Both the state-of-the-art and the proposed architectures were implemented in the same Virtex-7 FPGA running at 250MHz and using the same objective function, which allows for a direct comparison between the two. The proposed architecture achieves, for a dataset of the same size, a throughput

$3,02\times$ higher than the state-of-the-art implementation. Since the power of the state-of the-art-circuit is not specified, no comparison can be done.

The heterogeneous system referred in Table 5.11 and proposed in [56], uses the same FPGA architecture that is proposed in [22] to generate contingency tables. However, they are streamed to a GPU, where the objective function is calculated. This approach allows for the use of the total amount of resources in the FPGA for the generation of the contingency table, and, as it is calculated in a GPU, the use of a more complex objective function, which, in this case, is the information gain funtion.

Table 5.11: $K = 3$ results for a dataset containing 10 000 SNPs and 5000 patients

Architecture	Time	Comb/sec x 10^{-12}
Heterogeneous system[57]	1h 48min 31s	0.13
Virtex-7 690T [Proposed]	11min 24s	1.22

Streaming the contingency tables to the GPU has to be done through the Host computer which introduces a bottleneck. This is observable when comparing the results of the heterogeneous system with the FPGA-only implementation of the proposed architecture, targeting the same dataset (see Table 5.11). Although the used objective functions differ, the proposed architecture (implemented in the same FPGA as the heterogeneous system) running at the same 250MHz frequency, provides a throughput about $9.38\times$ higher when using a data set of the same size, which translates to a speedup of $9.7\times$.

The architectures can also run at different clock speeds, as verified in Section 5.1 by the implementations on the Zynq-7000 and the Zynq-Ultrascale+ SoCs, where the design run at 250MHz and 322MHz, respectively.

5.4 Summary

This chapter provided an in-depth evaluation in terms of FPGA resource utilization, power consumption, and performance, of different configurations of the proposed accelerator, targeting second, third, and fourth-order epistasis detection for various numbers of patients and SNPs. The accelerators were initially implemented for the ZYNQ7 Mini-ITX board, featuring a Zynq-7000 SoC. For particular numbers of patients, the generated architectures were also implemented in different FPGAs, to get a better sense of the improvements to the execution time enabled by implementing more CTUs and increasing the clock frequency.

On a second stage, the obtained results were compared against state-of-the-art FPGA and Heterogeneous (FPGA + GPU) implementations targeting second and third-order epistasis detection. The fourth-order architecture is not compared against any other implementation, at the time of writing, there is no other FPGA implementation targeting fourth-order epistasis detection based on exhaustive search.

Chapter 6

Conclusions

The increase in the computation complexity to perform high-order epistasis detection based on exhaustive search methods motivates the use of DSAs that can take advantage of the existent data parallelism, such as GPUs and FPGAs. However, the existent FPGA-based accelerators for epistasis detection, despite providing good results when comparing to CPU implementations, lack the flexibility to be used with any number of patients and the scalability to be adapted to higher-order epistasis, being limited to second, and third-order interactions. The proposed architecture addresses the limitations of the existent implementations by providing a general architecture that can be used to deploy FPGA-based accelerators for any order of interactions and targeting any number of patients. The increased performance and energy efficiency obtained when using the proposed second and third-order accelerators allow for the implementation of the first FPGA-based accelerator for fourth-order epistasis detection.

The proposed architecture was implemented for second, third, and fourth order interactions, targeting numbers of patients ranging from 64 to 8000 in the ZYNQ7-Mini ITX and Zynq-Ultrascale+ ZCU102 boards and in a Virtex-7 690T FPGA. When comparing the considered implementations of the proposed architecture with state-of-the art-FPGA implementations for second and third-order epistasis detection, it is clear that the proposed architectures provide $3\times$ faster execution times, for third-order epistasis detection when using similar hardware, and significantly lower energy consumption, when comparing against the state-of-the-art implementation targeting second-order interactions. Despite producing better performing architectures than the state-of-the-art, the main advantage of the method proposed in this thesis is the added flexibility that it provides by allowing for an easy generation of highly efficient architectures targeting any order of interactions and any dataset size.

6.1 Future Work Guidelines

The flexibility that is provided by the proposed architecture parameterization capabilities allows for the creation of efficient hardware architectures to perform exhaustive search epistasis detection of any order, with any number of patients, and implementable on any FPGA. The higher number of contingency table units obtained when implementing the generated architectures in a Virtex-7 690T prove that the

generated architectures are scalable with the size of the FPGA, producing better results the bigger the targeted FPGA is. If the generated architectures were to be implemented in a state-of-the-art platform, such as the Xilinx Versal Premium series, which can provide more than 3 million LUTs and more than 14000 DSP engines built on a 7nm technology, the number of implemented CTUs would increase and the design would be able to run at higher clock frequencies. The use of such a platform would greatly benefit the execution time, allowing for the use of bigger datasets for third and fourth-order epistasis detection, and even for the implementation of architectures targeting higher-order epistasis. The partial reconfiguration capabilities of modern FPGAs could also be exploited to further increase the flexibility of the proposed architecture, by allowing changes to the number of patients, and even to the adopted objective function in runtime.

The implementation of the generated architectures in an Application Specific Integrated Circuit (ASIC) would result in very significant power efficiency and performance improvements when compared to the FPGA-based implementations, as the architectures could run at higher clock frequencies, even if additional pipeline stages have to be added. When targeting an ASIC instead of an FPGA, the amount of implemented contingency table unit could also be significantly increased, since it would not be limited by the number of available FPGA resources.

Bibliography

[1] T. A. Manolio. Genomewide Association Studies and Assessment of the Risk of Disease. *New England Journal of Medicine*, 363(2):166–176, 2010. ISSN 0028-4793. doi: 10.1056/nejmra0905980.

[2] J. L. Haines, M. A. Hauser, S. Schmidt, W. K. Scott, L. M. Olson, P. Gallins, K. L. Spencer, S. Y. Kwan, M. Noureddine, J. R. Gilbert, N. Schnetz-Boutaud, A. Agarwal, E. A. Postel, and M. A. Pericak-Vance. Complement Factor H Variant Increases the Risk of Age-Related Macular Degeneration. *Science*, 308(5720):419 LP – 421, apr 2005. doi: 10.1126/science.1110359. URL http://science.sciencemag.org/content/308/5720/419.abstract.

[3] J. L. Vassy, M.-F. Hivert, B. Porneala, M. Dauriz, J. C. Florez, J. Dupuis, D. S. Siscovick, M. Fornage, L. J. Rasmussen-Torvik, C. Bouchard, and J. B. Meigs. Polygenic Type 2 Diabetes Prediction at the Limit of Common Variant Detection. *Diabetes*, 63:2172–2182, 2014. doi: 10.2337/db13-1663. URL http://diabetes.diabetesjournals.org/lookup/suppl/doi:10.2337/db13-1663/-/DC1.

[4] J. H. Moore, F. W. Asselbergs, and S. M. Williams. Bioinformatics challenges for genome-wide association studies. *BIOINFORMATICS REVIEW*, 26(4):445–455, 2010. doi: 10.1093/bioinformatics/btp713. URL https://academic.oup.com/bioinformatics/article/26/4/445/244836.

[5] W. H. Wei, G. Hemani, and C. S. Haley. Detecting epistasis in human complex traits. *Nature Reviews Genetics*, 15(11):722–733, 2014. ISSN 14710064. doi: 10.1038/nrg3747. URL http://dx.doi.org/10.1038/nrg3747.

[6] J. C. Turton, J. Bullock, C. Medway, H. Shi, K. Brown, O. Belbin, N. Kalsheker, M. M. Carrasquillo, D. W. Dickson, N. R. Graff-Radford, R. C. Petersen, S. G. Younkin, K. Morgan, and F. Panza. Investigating Statistical Epistasis in Complex Disorders. *Journal of Alzheimer's Disease*, 25:635–644, 2011. doi: 10.3233/JAD-2011-110197. URL http://www.broadinstitute.org/mpg/snap/ldsearch.php.

[7] T. D. Howard, G. H. Koppelman, A. Xu, S. L. Zheng, D. S. Postma, D. A. Meyers, and E. R. Bleecker. Gene-gene interaction in asthma: Il4ra and il13 in a dutch population with asthma. *American Journal of Human Genetics*, 70(1):230–236, 2002. ISSN 00029297. doi: 10.1086/338242.

[8] Y. M. Cho, M. D. Ritchie, J. H. Moore, J. Y. Park, K. U. Lee, H. D. Shin, H. K. Lee, and K. S. Park. Multifactor-dimensionality reduction shows a two-locus interaction associated with Type 2 diabetes

mellitus. *Diabetologia*, 47(3):549–554, 2004. ISSN 0012186X. doi: 10.1007/s00125-003-1321-3. URL http://www.wma.net/e/policy/17cnote.pdf.

[9] C. Niel, C. Sinoquet, C. Dina, and G. Rocheleau. A survey about methods dedicated to epistasis detection. *Frontiers in Genetics*, 6(SEP), 2015. ISSN 16648021. doi: 10.3389/fgene.2015.00285.

[10] T. F. Mackay and J. H. Moore. Why epistasis is important for tackling complex human disease genetics, jun 2014. ISSN 1756994X. URL http://genomemedicine.biomedcentral.com/articles/10.1186/gm561.

[11] X. Wan, C. Yang, Q. Yang, H. Xue, X. Fan, N. L. Tang, and W. Yu. BOOST: A fast approach to detecting gene-gene interactions in genome-wide case-control studies, 2010. ISSN 00029297.

[12] J. Piriyapongsa, C. Ngamphiw, A. Intarapanich, S. Kulawonganunchai, A. Assawamakin, C. Bootchai, P. J. Shaw, and S. Tongsima. iLOCi: a SNP interaction prioritization technique for detecting epistasis in genome-wide association studies. *BMC genomics*, 13 Suppl 7(Suppl 7), 2012. ISSN 14712164. doi: 10.1186/1471-2164-13-s7-s2.

[13] J. L. Hennessy and D. A. Patterson. A new golden age for computer architecture. *Communications of the ACM*, 62(2):48–60, feb 2019. ISSN 15577317. doi: 10.1145/3282307.

[14] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, and S. Wei. A High Energy Efficient Reconfigurable Hybrid Neural Network Processor for Deep Learning Applications. *IEEE Journal of Solid-State Circuits*, 53(4):968–982, apr 2018. ISSN 00189200. doi: 10.1109/JSSC.2017.2778281.

[15] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. GPU Computing. *Proceedings of the IEEE*, 96(5):879–899, may 2008. ISSN 1558-2256. doi: 10.1109/JPROC.2008.917757.

[16] R. Nobre, A. Ilic, S. Santander-Jimenez, and L. Sousa. Exploring the Binary Precision Capabilities of Tensor Cores for Epistasis Detection. *Proceedings - 2020 IEEE 34th International Parallel and Distributed Processing Symposium, IPDPS 2020*, pages 338–347, 2020. doi: 10.1109/IPDPS47924.2020.00043.

[17] J. González-Domínguez and B. Schmidt. GPU-accelerated exhaustive search for third-order epistatic interactions in case-control studies. *Journal of Computational Science*, 8:93–100, 2015. ISSN 18777503. doi: 10.1016/j.jocs.2015.04.001. URL http://dx.doi.org/10.1016/j.jocs.2015.04.001.

[18] G. Chrysos, E. Sotiriades, C. Rousopoulos, A. Dollas, A. Papadopoulos, I. Kirmitzoglou, V. Promponas, T. Theocharides, G. Petihakis, J. Lagnel, P. Vavylis, and G. Kotoulas. Opportunities from the use of FPGAs as platforms for bioinformatics algorithms. *IEEE 12th International Conference on BioInformatics and BioEngineering, BIBE 2012*, (November):559–565, 2012. doi: 10.1109/BIBE.2012.6399733.

[19] L. Wienbrandt. Bioinformatics applications on the FPGA-based high-performance computer RIVYERA. In *High-Performance Computing Using FPGAs*, volume 9781461417, pages 81–103. Springer New York, New York, NY, 2013. ISBN 9781461417910. doi: 10.1007/978-1-4614-1791-0_3.

[20] A. Surendar. FPGA based parallel computation techniques for bioinformatics applications. *International Journal of Research in Pharmaceutical Sciences*, 8(2):124–128, 2017. ISSN 09757538.

[21] L. Wienbrandt, J. C. Kässens, J. González-Domínguez, B. Schmidt, D. Ellinghaus, and M. Schimmler. FPGA-based acceleration of detecting statistical epistasis in GWAS. *Procedia Computer Science*, 29:220–230, 2014. ISSN 18770509. doi: 10.1016/j.procs.2014.05.020.

[22] J. C. Kässens, L. Wienbrandt, J. González-Domínguez, B. Schmidt, and M. Schimmler. High-speed exhaustive 3-locus interaction epistasis analysis on FPGAs. *Journal of Computational Science*, 9:131–136, jul 2015. ISSN 18777503. doi: 10.1016/j.jocs.2015.04.030. URL `https://www.sciencedirect.com/science/article/pii/S187775031500068X`.

[23] S. M. Trimberger. Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology: This Paper Reflects on How Moore's Law Has Driven the Design of FPGAs Through Three Epochs: The Age of Invention, the Age of Expansion, and the Age of Accumulation. *IEEE Solid-State Circuits Magazine*, 10(2):16–29, mar 2018. ISSN 19430582. doi: 10.1109/MSSC.2018.2822862.

[24] H. J. Cordell. Epistasis: what it means, what it doesn't mean, and statistical methods to detect it in humans. *Human Molecular Genetics*, 11(20):2463–2468, 2002. ISSN 0964-6906. doi: 10.1093/hmg/11.20.2463.

[25] L. Wienbrandt, J. C. Kässens, M. Hübenthal, and D. Ellinghaus. 1000× faster than PLINK: Combined FPGA and GPU accelerators for logistic regression-based detection of epistasis. *Journal of Computational Science*, 2019. ISSN 18777503. doi: 10.1016/j.jocs.2018.12.013.

[26] S. Purcell, B. Neale, K. Todd-Brown, L. Thomas, M. A. Ferreira, D. Bender, J. Maller, P. Sklar, P. I. De Bakker, M. J. Daly, and P. C. Sham. PLINK: A tool set for whole-genome association and population-based linkage analyses. *American Journal of Human Genetics*, 2007. ISSN 00029297. doi: 10.1086/519795.

[27] T. LaFramboise. Single nucleotide polymorphism arrays: A decade of biological, computational and technological advances. *Nucleic Acids Research*, 37(13):4181–4193, 2009. ISSN 03051048. doi: 10.1093/nar/gkp552. URL `www.ncbi.nlm.nih.gov/projects/SNP/`.

[28] A. Buniello, J. A. Macarthur, M. Cerezo, L. W. Harris, J. Hayhurst, C. Malangone, A. McMahon, J. Morales, E. Mountjoy, E. Sollis, D. Suveges, O. Vrousgou, P. L. Whetzel, R. Amode, J. A. Guillen, H. S. Riat, S. J. Trevanion, P. Hall, H. Junkins, P. Flicek, T. Burdett, L. A. Hindorff, F. Cunningham, and H. Parkinson. The NHGRI-EBI GWAS Catalog of published genome-wide association studies,

targeted arrays and summary statistics 2019. *Nucleic Acids Research*, 47(D1):D1005–D1012, 2019. ISSN 13624962. doi: 10.1093/nar/gky1120.

[29] K. Ozaki, Y. Ohnishi, A. Iida, A. Sekine, R. Yamada, T. Tsunoda, H. Sato, H. Sato, M. Hori, Y. Nakamura, and T. Tanaka. Functional SNPs in the lymphotoxin-α gene that are associated with susceptibility to myocardial infarction. *Nature Genetics*, 32(4):650–654, 2002. ISSN 1546-1718. doi: 10.1038/ng1047. URL https://doi.org/10.1038/ng1047.

[30] R. J. Klein, C. Zeiss, E. Y. Chew, J. Y. Tsai, R. S. Sackler, C. Haynes, A. K. Henning, J. P. SanGiovanni, S. M. Mane, S. T. Mayne, M. B. Bracken, F. L. Ferris, J. Ott, C. Barnstable, and J. Hoh. Complement factor H polymorphism in age-related macular degeneration. *Science*, 308(5720):385–389, apr 2005. ISSN 00368075. doi: 10.1126/science.1109557. URL /pmc/articles/PMC1512523/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC1512523/.

[31] W. Bateson. Mendel's Principles of Heredity. *Nature*, 86(2169):407, 1911. ISSN 1476-4687. doi: 10.1038/086407a0. URL https://doi.org/10.1038/086407a0.

[32] R. A. Fisher. XV.—The Correlation between Relatives on the Supposition of Mendelian Inheritance. *Transactions of the Royal Society of Edinburgh*, 52(2):399–433, 1919. ISSN 0080-4568. doi: DOI:10.1017/S0080456800012163. URL https://www.cambridge.org/core/article/xvthe-correlation-between-relatives-on-the-supposition-of-mendelian-inheritance/A60675052E0FB78C561F66C670BC75DE.

[33] M. Robnik-Šikonja and I. Kononenko. Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning*, 53(1-2):23–69, oct 2003. ISSN 08856125. doi: 10.1023/A:1025667309714.

[34] B. J. Grady, E. S. Torstenson, P. J. Mclaren, P. I. W. De Bakker, D. W. Haas, G. K. Robbins, R. M. Gulick, R. Haubrich, H. Ribaudo, and M. D. Ritchie. USE OF BIOLOGICAL KNOWLEDGE TO INFORM THE ANALYSIS OF GENE-GENE INTERACTIONS INVOLVED IN MODULATING VIROLOGIC FAILURE WITH EFAVIRENZ-CONTAINING TREATMENT REGIMENS IN ART-NAÏVE ACTG CLINICAL TRIALS PARTICIPANTS NIH Public Access. Technical report, 2011.

[35] R. Jiang, W. Tang, X. Wu, and W. Fu. A random forest approach to the detection of epistatic interactions in case-control studies. 2009. doi: 10.1186/1471-2105-10-S1-S65. URL http://www.biomedcentral.com/1471-2105/10/S1/S65.

[36] B. Han, X. wen Chen, Z. Talebizadeh, and H. Xu. Genetic studies of complex human diseases: Characterizing SNP-disease associations using Bayesian networks. *BMC Systems Biology*, 6(SUPPL3), 2012. ISSN 17520509. doi: 10.1186/1752-0509-6-S3-S14. URL http://www.biomedcentral.com/1752-0509/6/S3/S14.

[37] Y. Wang, X. Liu, K. Robbins, and R. Rekaya. Open Access TECHNICAL NOTE AntEpiSeeker: detecting epistatic interactions for case-control studies using a two-stage ant colony optimization algorithm. Technical report, 2010. URL http://www.biomedcentral.com/1756-0500/3/117.

[38] R. J. Tallarida and R. B. Murray. Chi-Square Test BT - Manual of Pharmacologic Calculations: With Computer Programs. pages 140–142. Springer New York, New York, NY, 1987. ISBN 978-1-4612-4974-0. doi: 10.1007/978-1-4612-4974-0_43. URL https://doi.org/10.1007/978-1-4612-4974-0{_}43.

[39] Y. Sun, J. Shang, J. X. Liu, S. Li, and C. H. Zheng. EpiACO - A method for identifying epistasis based on ant Colony optimization algorithm. *BioData Mining*, 10(1), 2017. ISSN 17560381. doi: 10.1186/s13040-017-0143-7. URL https://sourceforge.net/projects/.

[40] P. G. Ferrario and I. R. Kö. Transferring entropy to the realm of GxG interactions. doi: 10.1093/bib/bbw086. URL https://academic.oup.com/bib/article-abstract/19/1/136/2566836.

[41] S. Leem, H.-h. Jeong, J. Lee, K. Wee, and K.-A. Sohn. Fast detection of high-order epistatic interactions in genome-wide association studies using information theoretic measure. *Computational biology and chemistry*, 50:19–28, jun 2014. ISSN 1476-928X (Electronic). doi: 10.1016/j.compbiolchem.2014.01.005.

[42] T. Hu, Y. Chen, J. W. Kiralis, R. L. Collins, C. Wejse, G. Sirugo, S. M. Williams, and J. H. Moore. An information-gain approach to detecting three-way epistatic interactions in genetic association studies. *Journal of the American Medical Informatics Association*, 20(4):630–636, 2013. ISSN 10675027. doi: 10.1136/amiajnl-2012-001525.

[43] B. Goudey, D. Rawlinson, Q. Wang, F. Shi, H. Ferra, R. M. Campbell, L. Stern, M. T. Inouye, C. S. Ong, and A. Kowalczyk. GWIS–model-free, fast and exhaustive search for epistatic interactions in case-control GWAS. *BMC genomics*, 14 Suppl 3(Suppl 3):1–18, 2013. ISSN 14712164. doi: 10.1186/1471-2164-14-s3-s10.

[44] X. Li. A fast and exhaustive method for heterogeneity and epistasis analysis based on multi-objective optimization. *Bioinformatics*, 33(18):2829–2836, 2017. ISSN 14602059. doi: 10.1093/bioinformatics/btx339.

[45] X. Cao, G. Yu, J. Liu, L. Jia, and J. Wang. ClusterMI: Detecting high-order SNP interactions based on clustering and mutual information. *International Journal of Molecular Sciences*, 19(8), 2018. ISSN 14220067. doi: 10.3390/ijms19082267. URL www.mdpi.com/journal/ijms.

[46] J. Pensar, S. Puranen, B. Arnold, N. MacAlasdair, J. Kuronen, G. Tonkin-Hill, M. Pesonen, Y. Xu, A. Sipola, L. Sánchez-Busó, J. A. Lees, C. Chewapreecha, S. D. Bentley, S. R. Harris, J. Parkhill, N. J. Croucher, and J. Corander. Genome-wide epistasis and co-selection study using mutual information. *Nucleic Acids Research*, 47(18):e112–e112, oct 2019. ISSN 0305-1048. doi: 10.1093/nar/gkz656. URL https://doi.org/10.1093/nar/gkz656.

[47] J. Gonzalez-Dominguez, L. Wienbrandt, J. C. Kaassens, D. Ellinghaus, M. Schimmler, and B. Schmidt. Parallelizing epistasis detection in GWAS on FPGA and GPU-accelerated computing systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 12(5):982–994, 2015. ISSN 15455963. doi: 10.1109/TCBB.2015.2389958.

[48] C. Ponte-Fernández, J. González-Domínguez, and M. J. Martín. Fast search of third-order epistatic interactions on CPU and GPU clusters. *International Journal of High Performance Computing Applications*, 34(1):20–29, 2020. ISSN 17412846. doi: 10.1177/1094342019852128. URL `https://github.com/chponte/mpi3snp`.

[49] J. Burgess. RTX on—The NVIDIA Turing GPU. *IEEE Micro*, 40(2):36–44, 2020. ISSN 1937-4143 VO - 40. doi: 10.1109/MM.2020.2971677.

[50] L. Wienbrandt. Bioinformatics Applications on the FPGA-Based High-Performance Computer RIVYERA BT - High-Performance Computing Using FPGAs. pages 81–103. Springer New York, New York, NY, 2013. ISBN 978-1-4614-1791-0. doi: 10.1007/978-1-4614-1791-0_3. URL `https://doi.org/10.1007/978-1-4614-1791-0{_}3`.

[51] A. Mahram and M. C. Herbordt. NCBI BLASTP on high-performance reconfigurable computing systems. *ACM Transactions on Reconfigurable Technology and Systems*, 7(4), 2015. ISSN 19367414. doi: 10.1145/2629691.

[52] S. A., A. M., and S. P. P. A parallel reconfigurable platform for efficient sequence alignment. *African Journal of Biotechnology*, 13(33):3344–3351, 2014. ISSN 1684-5315. doi: 10.5897/ajb2014.13680.

[53] N. Neves, N. Sebastião, A. Patricio, D. Matos, P. Tomás, P. Flores, and N. Roma. BioBlaze: Multi-core SIMD ASIP for DNA sequence alignment. In *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, pages 241–244, 2013. ISBN 2160-052X VO -. doi: 10.1109/ASAP.2013.6567581.

[54] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan. FPGA implementation of K-means algorithm for bioinformatics application: An accelerated approach to clustering Microarray data. In *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 248–255, 2011. ISBN VO -. doi: 10.1109/AHS.2011.5963944.

[55] M. Smerdis, P. Dagritzikos, G. Chrysos, E. Sotiriades, and A. Dollas. Reconfigurable Systems for the Zuker and Predator Algorithms for Secondary Structure Prediction of Genetic Data. In *2010 International Conference on Field Programmable Logic and Applications*, pages 448–451, 2010. ISBN 1946-1488 VO -. doi: 10.1109/FPL.2010.91.

[56] J. C. Kassens, L. Wienbrandt, M. Schimmler, J. Gonzalez-Dominguez, and B. Schmidt. Combining GPU and FPGA technology for efficient exhaustive interaction analysis in GWAS. *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, 2016-Novem:170–175, 2016. ISSN 10636862. doi: 10.1109/ASAP.2016.7760788.

[57] L. Wienbrandt, J. C. Kassens, M. Hübenthal, and D. Ellinghaus. Fast Genome-Wide Third-order SNP Interaction Tests with Information Gain on a Low-cost Heterogeneous Parallel FPGA-GPU Computing Architecture. In *Procedia Computer Science*, 2017. doi: 10.1016/j.procs.2017.05.210.

[58] Avnet. Xilinx Zynq ® -7000 All Programmable SoC Mini-ITX Development Kit Getting Started Guide. pages 1–14, 2017.

[59] Xilinx. Zynq-7000 SoC Technical Reference Manual. *Ug585*, 585:1–1843, 2018.

[60] Xilinx. ZCU102 Evaluation. 1182:1–120, 2018.

[61] Xilinx. Zynq UltraScale+ Device Technical Reference Manual. *Electronics: Science, Technology, Business*, 195(4):64–67, 2020. ISSN 1992-4178. doi: 10.22184/1992-4178.2020.195.4.64.67.

[62] Xilinx. DSP48E1 Slice User Guide. *Xilinx*, 479:1–56, 2017.

[63] Xilinx. Xilinx Adder/Subtracter. *LogiCORE IP Product Guide*, 2015.

[64] Xilinx. Xilinx Floating-Point Operator. 2020.