electronics

MDPI

Article

Integrating FPGA Acceleration in the DNAssim Framework for Faster DNA-Based Data Storage Simulations

Alessia Marelli [1,*,†], Thomas Chiozzi [1,†], Nicholas Battistini [1,†], Lorenzo Zuolo [1,†], Rino Micheloni [1,†] and Cristian Zambelli [2,*,†]

[1] DNAalgo, 62100 Macerata, Italy
[2] Dipartimento di Ingegneria, Università degli Studi di Ferrara, 44122 Ferrara, Italy
* Correspondence: alessia.marelli@dnaalgo.com (A.M.); cristian.zambelli@unife.it (C.Z.); Tel.: +39-320-2472597 (A.M.); +39-0532-974993 (C.Z.)
† These authors contributed equally to this work.

Abstract: DNA-based data storage emerged in this decade as a promising solution for long data durability, low power consumption, and high density. However, such technology has not yet reached a good maturity level, requiring many investigations to improve the information encoding and decoding processes. Simulations can be key to overcoming the time and the cost burdens of the many experiments imposed by thorough design space explorations. In response to this, we have developed a DNA storage simulator (DNAssim) that allows simulating the different steps in the DNA storage pipeline using a proprietary software infrastructure written in Python/C language. Among the many operations performed by the tool, the edit distance calculation used during clustering operations has been identified as the most computationally intensive task in software, thus calling for hardware acceleration. In this work, we demonstrate the integration in the DNAssim framework of a dedicated FPGA hardware accelerator based on the Xilinx VC707 evaluation kit to boost edit distance calculations by up to 11 times with respect to a pure software approach. This materializes in a clustering simulation latency reduction of up to 5.5 times and paves the way for future scale-out DNA storage simulation platforms.

Keywords: DNA-based storage; FPGA hardware acceleration; simulation

Citation: Marelli, A.; Chiozzi, T.; Battistini, N.; Zuolo, L.; Micheloni, R.; Zambelli, C. Integrating FPGA Acceleration in the DNAssim Framework for Faster DNA-Based Data Storage Simulations. *Electronics* **2023**, *12*, 2621. https://doi.org/10.3390/electronics12122621

Academic Editor: Gemma Piella

Received: 27 April 2023
Revised: 23 May 2023
Accepted: 2 June 2023
Published: 10 June 2023

1. Introduction

The expected amount of data generated over the next decade will rapidly expose the need for larger scale-out file- and object-based storage [1]. One of the main culprits in the forecast storage emergency is the set of processes that cloud-based infrastructures devise nowadays to preserve the information while attempting to keep the Quality of Service (QoS) within the limits of a Service-Level Agreement (SLA) [2]. Frequent data replication and retention management require denser, faster, and yet more reliable storage media, impacting the Total Cost of Ownership (TCO). As a result, there is a surge of interest in exploring alternatives to current storage technologies, such as Solid State Drives (SSDs), Hard Disk Drives (HDDs), and magnetic tapes. An interesting solution to reduce TCO and face the challenges in the cloud archival tier emerged in deoxyribonucleic acid (DNA)-based data storage [3]. In a nutshell, DNA storage refers to the ability to represent digital information (i.e., files made of '1' and '0' bits) in a synthetic molecule composed of two polymer chains containing a sequential string of nucleotide monomers (bases). The chains form a double helix structure in which there are four naturally occurring DNA bases: adenine (A), thymine (T), cytosine (C), and guanine (G) [4]. From a digital standpoint, we could, for example, use Gray coding to assign the value 11 to "A", 10 to "T", 00 to "C" and 01 to "G" (see Figure 1). The data stored in DNA would then last reliably for thousands of years [5] with little power consumption requirements for retention refreshing. Further, a storage density of about 100 PB per gram is predicted [6], with evident benefits on the physical footprint of a storage system. The molecules that represent the information

to be stored in DNA are created on demand by encoding the way the DNA molecule is synthesized and assembled. Once available, DNA sequences are later read back using proper decoding techniques and sequencing technologies. However, one of the aspects that limit the research in DNA-based data storage is related to economic reasons. Today, writing (synthesis) and reading (sequencing) DNA for data storage are not practical at scale [3]. Synthesis costs for DNA data storage are dependent on how bits are encoded into DNA bases, and on the specific methods of synthesizing the DNA. They are hard to characterize since today's applications do not include DNA data storage (despite a lab-based proof of concept [7]), but in general are considered unaffordable for thorough design space explorations. Sequencing costs have already dropped dramatically in this decade [8,9], but are still too high to bear since we are in the phase of gaining knowledge on the DNA decoding steps for storage, thus requiring a multitude of lab experiments being performed to explore the plethora of parameters controlling the process. Additionally, the time required for synthesis and sequencing (with the former dominating the experiment time) of DNA to prove reliable storage capabilities is extremely high, being in the range of a week for less than 5 kbits [10]).

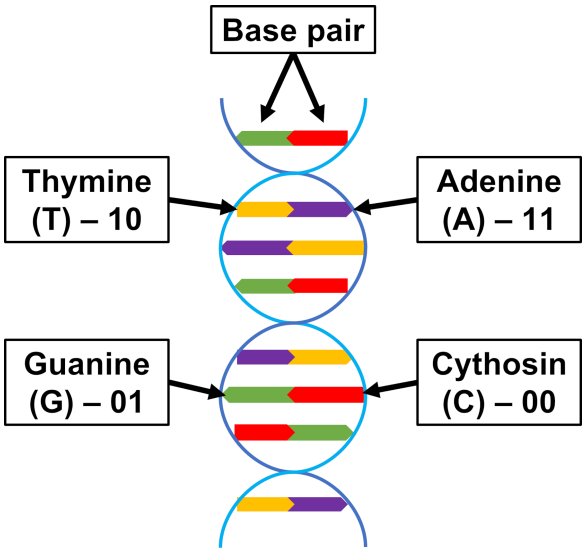

Figure 1. The double helix structure of the DNA with the base pairs evidenced along with the digital encoding of the bases.

On top of that, the DNA synthesis and sequencing processes in information encoding and decoding are inherently error-prone (i.e., noise affected). Common errors found in each DNA chain (strand) are insertions, deletions, substitutions of bases, and multiple "noisy" amplified replicas when the polymerase chain reaction (PCR) is used in data readout [11]. The nature of encoding and decoding errors is purely stochastic [12] and tailored to the specific technology exploited in DNA synthesis/sequencing. Moreover, since the DNA data storage channel theory observed some undefined spots in the phenomenology of errors [13], there are some inherent difficulties in projecting their impact at the storage application level. The literature addressed these specific topics by focusing either on a deeper characterization of the channel or on the error correction techniques and on random access data retrieval [5,12,14–19], although the difficulty of understanding how errors interact in the digital data-to-DNA pipeline [3] still persists. A thorough campaign of experiments could help in the formulation of error models, but with an evident burden on the implementation cost and on the time-to-results. This calls for the creation of ad hoc DNA-based storage simulation environments that tackle all the aspects of the synthesis and sequencing errors.

In this work, we address the challenges described so far by providing the following major contributions in the field of DNA-based storage:

- We have developed a DNA storage simulation (DNAssim) platform to enable a fast full-design exploration of the synthesis and sequencing technologies in the context of storing digital information inside a DNA strand. The tool is entirely built in the Python/C language and features a proprietary Graphical User Interface (GUI). To the best of our knowledge, this is the first framework oriented to the study of DNA-based storage that includes all the steps of the pipeline within a single tool.
- In the storage pipeline simulation, we identified a possible performance bottleneck in the calculation of the edit distance, which is a similarity metric between DNA strands appearing both in the modeling of the DNA storage noise channel and in the information decoding steps.
- To this extent, we developed a custom acceleration engine based on a Xilinx VC707 Field Programmable Gate Array (FPGA) that improves edit distance execution with evident advantages in the simulation chain. The accelerator improves the performance with respect to a software counterpart by up to 11 times (700 kedit/s) and consumes up to 7.46 W with a clock frequency of 170 MHz for the computational blocks.
- The accelerator has been integrated with the DNAssim framework by developing a custom driver in the C language that provides data conversion from the software tool and transfers them to the FPGA for subsequent edit distance calculation using the PCIe gen2 protocol [20].
- We have validated the hardware-software co-simulation approach in the clustering operation performed during the DNA storage decoding steps. The experimental results demonstrated a simulation latency reduction of up 5.5 times with respect to a pure software approach. Further, we have projected the simulation speed-ups achievable on real use cases by demonstrating a simulation time reduction of up to 4.2 times when considering the storage of a music file on the DNA.

This work largely extends the preliminary ideas provided in our previous presentations [21,22] where we disclosed the idea of having software such as DNAssim and its potential connection to an external hardware accelerator that had an unoptimized implementation. No details about the internal structure of the overall framework, the FPGA used, the driver interconnection with DNAssim, and the structure of the design with the obtained performance were given there. In this work, we completely exposed the details of the DNAssim platform by showing its potential in modeling the DNA channels for information reconstruction simulation and the architecture of the accelerator with quantitative and qualitative benchmarks to assess the benefits in some critical operations such as clustering (were edit distance is required the most).

With our work, we want to provide a simulation framework that will be of practical use for storage engineers that want to quickly understand the potentialities and the limitations of DNA-based data storage by exploring the role of the different steps in the storage pipeline without bearing the costs and the time of synthesis and sequencing. Considering the end-to-end flow from the information encoding to the final data reconstruction will help extrapolate the operation bottlenecks and the potential data read/write failed strategies. With our tool, it will be possible, without incurring the costs of synthesis/sequencing, to extract the information required in benchmarking the reliability of this new storage medium with respect to legacy devices such as Flash memories for Solid State Drives. Further, our co-simulation approach based on an FPGA accelerator addresses the shortcomings of computationally intensive simulation campaigns [23] by reducing the time-to-result for DNA storage data reconstruction using defined encoding/decoding strategies. Given the scalability of this methodology, we foresee the future integration of multiple accelerators dedicated to different steps of the storage pipeline enabling large scale-out simulation campaigns, with the ultimate goals of improving the confidence in DNA-based data storage and helping the research community in this context.

The work is organized as follows: in Section 2, we present the related works on the topic; in Section 3 we discuss the fundamentals of DNA-based storage and the DNAssim software simulation chain; in Section 4 we present the FPGA-based edit distance hardware

acceleration block to be integrated into the DNAssim framework; in Section 5 we provide experiments and results to assess the benefits of our approach in clustering operations performed in the information decoding pipeline; conclusions will be drawn in Section 6.

2. Related Works

DNA-based storage has been studied for more than one decade both from the theoretical and the practical standpoint. A recent proof-of-concept showing an automated system for DNA reading and writing has been provided in [24]. Microsoft and the University of Washington research teams created a device that takes the data input and then encodes it as DNA strands. To read it, a dedicated machine pumps the DNA strands into a DNA sequencer, and then a computer decodes that into binary to recreate the original data. This promising work enabled computer architects to start considering DNA storage technology even as an integral part of computer design [7]. With the help of the coding theory, there was an incredible effort put into the ECC application and the technology used for DNA strands' readout.

Y. Erlich et al. [5] use Fountain codes and Reed–Solomon (RS) codes to recover missing DNA strands from sequencing and errors such as substitutions within each DNA readout. They do not cope with errors such as insertion and deletions.

L. Organick et al. [18] also use RS codes, but on blocks with greater lengths than [5], to correct both erasures and substitution errors. They also introduce a random-access technique to the stored data based on the PCR. In this work, they also suggest interleaving the input data to make sure that the DNA strands obtained by synthesis are dissimilar. In this way, the decoding process becomes less expensive.

S. Chandak et al. [12] propose to use Low-Density Parity Check (LDPC) codes to face the problems of missing strands and substitution errors. They also suggest the use of a synchronization marker to solve the issues derived from insertion and deletion errors. During the decoding process, differently from [18], they cluster the reads by indexing due to its lower computational complexity. This is possible because during encoding they add an index to each strand.

C. Rashtchian et al. [23] highlight that DNA-based data storage requires a computationally intensive process to retrieve the data. A crucial step in the data retrieval pipeline involves the clustering of billions of strings. Their proposed algorithm iteratively merges clusters based on random representatives and compares only a small subset of representatives thanks to a hashing scheme that determines if the clusters are to be merged.

S. Yazdi et al. [17] introduce a DNA storage system with random access capabilities based on PCR. In this work, they demonstrate that the cost of synthesizing uncompressed files is much higher than the cost of compressing a file and adding redundancy to eliminate errors. An alignment procedure that compares with [25] is proposed to achieve high-quality reads to reduce the number of errors.

R. Heckel et al. [19] discuss the statistical characterization of the DNA storage channel. They describe the error sources of the channel by pinpointing three phases: during the DNA synthesis process, in the storage phase, and during the DNA sequencing process. More generally, they assert that most deletion and insertion errors are due to the synthesis, while the substitution errors are dominated by synthesis/sequencing and are also impacted by the DNA decay and the PCR.

S. R. Srinivasavaradhan et al. [26] present a new algorithm useful for trace reconstruction called Trellis-BMA. They also give a description of the DNA storage system channel and describe the channel considering that the error profile is strongly influenced by the DNA sequencing technology.

All these works focus on specific steps of the DNA storage pipeline and do not consider the end-to-end chain of operations required to encode and decode the information. Further, to the best of our knowledge, there is no report of a dedicated simulation engine applied to the context of DNA-based data storage.

Concerning FPGA acceleration engines exploited in speeding up the simulation pipeline and enabling co-simulation platforms, there has been a lot of investigation.

In [27], we preliminarily developed a co-simulation platform that exploits an FPGA for ECC analysis and error-floor characterization tightly coupled with a dedicated storage simulator [28]. The benefits of the co-simulation materialized in accurate yet time-effective predictions of the storage media reliability. Motivated by this result, we decided to study a similar approach in the domain of DNA storage.

In the specific context of hardware acceleration for DNA strand analysis, Caffarena et al. [29], Kent et al. [30] and Dydel et al. [31] present an FPGA hardware support that leverages parallel computation acceleration for specific functions of the DNA reconstruction chain, although in these papers, the computations are not fully on par with the current technologies available for DNA storage, but rather for biological analysis of the DNA.

Castells-Rufas et al. [32] developed an approximate edit distance accelerator validated on several up-to-date FPGA architectures (with PCIe gen3 $\times$16 host interface) reporting top-class performance in the context of DNA string alignment, although not providing a direct link with the clustering algorithm nor with DNA-based storage simulation acceleration.

Marchisio et al. [33] observed that since DNA strings suffer from variations such as mutation, noisy sampling, and transmission, instead of searching for the exact match, the inexact string matching (ISM) of DNA sequences is preferred. They show that due to the large amount of data and massive data dependency, the ISM algorithm is not suitable for being implemented into general-purpose hardware. They propose a novel specialized hardware architecture implemented on a Xilinx Ultrascale+ FPGA showing massive clock cycle reduction compared to an ARM-based implementation, although still with no direct link with DNA-based data storage.

In this work, we are trying to address what is still an unsolved part of the problem in DNA-based data storage, namely the creation of a time-efficient framework based on a hardware/software co-simulation environment that encompasses a tool for the complete design space exploration of the storage pipeline and a custom accelerator to boost the execution of computationally intensive tasks. This will be particularly important in large simulation campaigns required to expose the bottlenecks in the different stages of the DNA information encoding/decoding processes and in evidencing the critical aspects such as storage reliability (i.e., resilience to errors).

3. The DNA-Based Storage Simulation Engine

The simulation of the DNA storage pipeline involves different stages that should be modeled by a software framework capable of capturing the peculiarities of the information encoding and decoding. Figure 2 shows a high-level description of the proposed DNAssim software simulation engine. The tool has been written in Python version 3.10 with C libraries for accelerating the computation of time-intensive and CPU resource-demanding operations. It currently supports both single-thread and multi-thread architectures.

3.1. Encoder Blocks

The information encoding process starts by feeding the first block of the simulator, namely the *Outer code*, with arbitrary input data to store by using synthetic DNA (e.g., a multimedia file, a text file, etc.). Its role is to apply an ECC to the binary data in the input, thus granting data corruption protection against erasures or insertions/deletions (Indel channel analysis [34]). Supported ECCs range from RS to algebraic codes such as Bose–Chaudhuri–Hocquenghem (BCH) or probabilistic ones such as LDPC. It is worth noting that since we are in the presence of burst errors in the channel, a set of interleaving techniques typical for erasure coding are exploited [35].

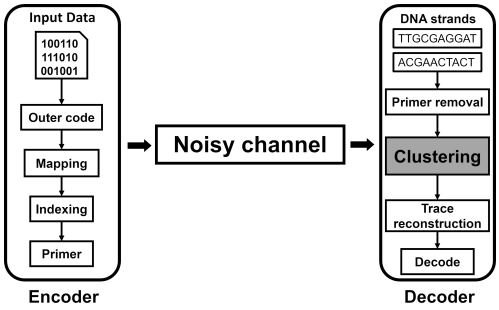

Figure 2. The architecture of the DNAssim software simulation engine presenting the different steps to be performed in the DNA storage pipeline.

The next operation to be performed during encoding is the so-called *Mapping*. The generated binary data from the *Outer code* are translated into bases (A, C, G, and T) by using a user-specified scheme of 2 bits (e.g., 00 — A, 01 — C, 10 — G, and 11 — T). Since [18] pointed out that during the mapping process we may incur the generation of homopolymers (e.g., AA couples) that induce synthesis errors, we included the application of a randomization scheme (with different techniques [36]) to improve the synthesis quality, as suggested by [12]. A synchronization marker inside the DNA strand can be applied to recover specific parts of the information in case of insertion and deletion errors.

The DNA strands are, however, sequenced without a specific order. DNA storage systems typically encode a single file into a pool of short strands to reduce synthesis and sequencing issues. To this extent, in order to reconstruct the original stored file, an ECC-protected index must be applied by an *Indexing block* [37]. Since DNA sequences with long homopolymer sequences result in issues with synthesis/sequencing [5], we also support pseudo-random permutations to each index before encoding. This approach proposed in [12] turns out to be effective in sequencing error reduction.

The last operation to simulate in the encoding process is the *Primer* application. For biological reasons, the primer is used not only for the synthesis process but also for sequencing the data to pick up the correct DNA strands that are to be read afterwards. In fact, it is possible to have multiple files stored in a pool of DNA strands.

3.2. Noise Model of the Channel

The observed noise in DNA storage is a complex combination of synthesis errors, amplification errors generated during the data readout by the PCR, and sequencing-induced noise [13,38]. In fact, the DNA is accessed using sequencing technologies, which results in several PCR noisy replicas called reads (see Figure 3). A read is a copy of an original short string of DNA symbols called a reference [23]. Both references and reads contain hundreds of symbols, forecasted to reach a thousand in the near future [17]. Symbols (bases) in DNA are inserted, deleted, and transposed/substituted (IDS) because of the previous operations. The DNA storage channel is, therefore, modeled as IDS-like, focusing on the case where a single encoded message is transmitted and multiple independent traces are observed [6,17,18]. However, a precise noise model of this error profile is found cumbersome and of impractical application [26]. A peculiarity of the DNA channel is that the input is a set of many strings with similar lengths mapped in the $\{A, T, C, G\}$ alphabet. Those strings may have a certain degree of similarity, namely, they feature a proper distance. For DNA-based storage, the similarity index is calculated with the edit distance, also defined as the Levenshtein distance [39], when IDS errors are present in the strings. The Wagner–Fischer algorithm [40] computes the edit distance between two strings based on the observation that if a matrix D holds the edit distances between all the symbols in the first string and all the symbols in the second, we can compute the values in D by a flood-filling procedure, and thus find the edit distance between the two full strings as the last value computed by the algorithm. Let us assume two strings x and y of N and M length, respectively. The D matrix is then sized $(N+1) \times (M+1)$. The algorithm is then

expected to calculate each element of the distance score matrix $D[i,j]$ ($i = 1, \cdots, N$ and $j = 1, \cdots, M$) using the following equation set:

$$\begin{aligned} D[i,0] &= i \qquad i = 1, \cdots, N \\ D[0,j] &= j \qquad j = 1, \cdots, M \end{aligned} \tag{1}$$

$$s = \begin{cases} 0 & x_i = y_j \\ 1 & x_i \neq y_j \end{cases} \tag{2}$$

$$D[i,j] = min \begin{pmatrix} D[i-1,j-1] + s \\ D[i-1,j] + 1 \\ D[i,j-1] + 1 \end{pmatrix} \begin{matrix} i = 1, \cdots, N \\ j = 1, \cdots, M \end{matrix} \tag{3}$$

where the terms $D[i-1,j-1]+s$, $D[i-1,j]+1$, and $D[i,j-1]+1$ in Equation (3) account for symbol substitutions, deletions, and insertions. We remember that if one of the two strings has zero length, the result is the length of the non-null string. If both strings have zero length, the result is zero. An example of edit distance calculation through the matrix D used by the Wagner–Fischer algorithm is provided in Figure 4. To evaluate the noise-modeling capabilities of the DNAssim software engine, we simulated the storage of a 2 MB file. We fixed an IDS error rate equal to 1%. In Figure 5a, there is an example of three different PCR profiles that DNAssim is able to reproduce. It represents the probability density function (PDF) as a function of the mean coverage. We define as mean coverage the average number of reads per strand. An effective way to decrease the IDS error rate and the number of erasures in DNA-based storage is to increase the mean coverage. A high coverage means that the same strand is read many times, so it is less likely to feature a missing strand (i.e., an erasure). Further, the number of strands used to produce a consensus is higher, so it is most likely to have a good consensus for statistical reasons. Deviations from the average coverage in the sequence copy distribution can either cause wasteful provisioning in sequencing or an excessive number of missing sequences.

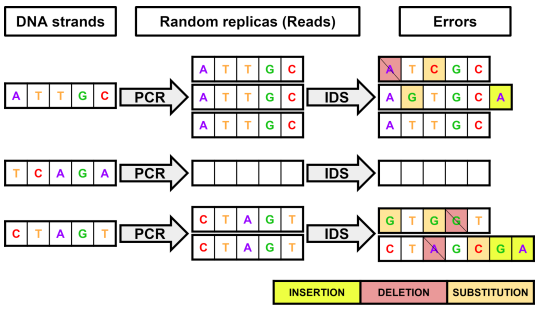

Figure 3. Common PCR-induced and IDS channel errors modeled by the DNAssim software engine.

		A	C	C	T	G
	0	1	2	3	4	5
T	1	1	2	3	3	4
G	2	2	2	3	4	3
G	3	3	3	3	4	4
A	4	3	4	4	4	5
T	5	4	4	5	4	5
C	6	5	4	4	5	5

$S = 1$
$D[4,3] = \min(3+1, 4, 5)$

Figure 4. An example of edit distance calculation through the Wagner–Fischer algorithm using a score matrix D. The value highlighted in green (i.e., the last element of the matrix) is the edit distance between the two strings.

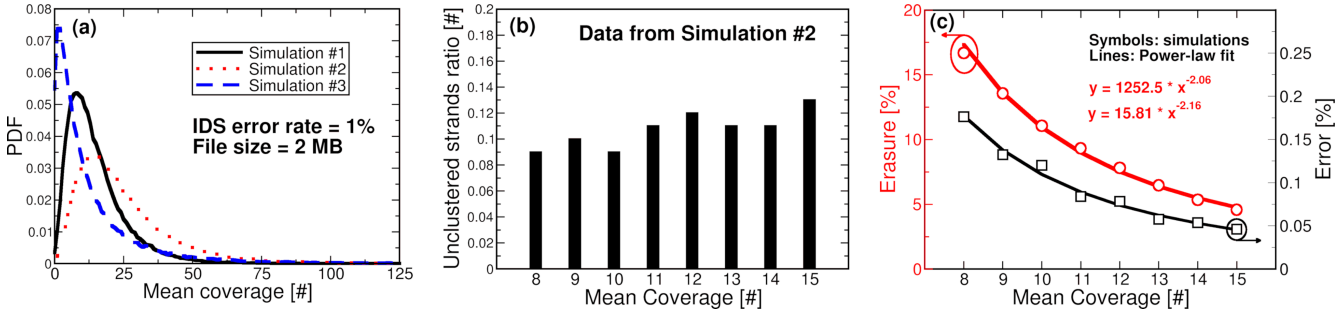

Figure 5. (**a**) PDF of the mean coverage extracted from simulations where three different PCR profiles are considered. (**b**) Unclustered strands ratio as a function of the mean coverage extracted from a simulation profile. (**c**) Erasure and error percentage sustained by the decoder as a function of the mean coverage.

Tuning our PCR models, we are able to represent a case where the mean coverage is low, but it exposes a long tail in the PDF, a case where the mean coverage is slightly higher while the tail has approximately the same length, and another simulation where the mean coverage is much higher, turning the PDF to a Gaussian-like shape.

3.3. Decoder Blocks

The simulation of the decoding operation starts with the *Primer removal* from the DNA strands. Since the PCR operation induces many read replicas, the goal of the decoder is to recover the reference strings (the stored data) from the observed reads. This operation is accomplished by the *Clustering*. Datasets in DNA-based storage typically contain only a handful of reads for each reference, and each of these reads might differ depending on IDS errors introduced by the channel [23]. The challenge of clustering is to achieve high precision and recall metrics of many small underlying clusters. In DNA storage, the cluster size can range from one up to more than a hundred noisy copies of the same reference and all the clusters are separated in terms of edit distance [23]. Since the calculations of this operation can be computationally intensive, we need to minimize the number of edit distance evaluations (calculated through the Wagner–Fischer algorithm discussed in the former section) through a carefully chosen hash function. The hash function $H(w,l)$ is defined as the enumeration of all strings w with length l in the alphabet $\{A,T,C,G\}$ (e.g., $AAAA$, $AAAC$, $AACC$, etc.). A member $h(w,l)$ of the family $H(w,l)$ is a random permutation of that enumeration. To hash a cluster C with $h(w,l)$, the following steps are required:

- Pick a random element (a string) of the cluster and call it x;
- Evaluate which sequence in the enumeration $h(w,l)$ appears first in x;
- Return that sequence followed by the next l character of x after the sequence.

As an example, suppose that we have a long string $x = AACTAGCTTAGCAAGT$ as a random member of the cluster C. Suppose the enumeration of $h(4,5)$ is $AATT$, $GGAC$, $GTAC$, $GCTT$, etc. Then, the hash of x is $GCTTAGCAA$. Algorithm 1 shows how clustering is performed inside DNAssim.

Algorithm 1 Clustering algorithm

Ensure: C, r, w, l, L
  for all $l \in 1 \cdots L$ **do**
    pick random permutation $h(w,l) \in H(w,l)$
    for all clusters c $\in C$ **do**
      pick random element x_c and hash it
      for all pairs $x, y : h(w,l)(x) == h(w,l)(y)$ **do**
        if $edit\_distance(x,y) \leq r$ **then**
          merge C_x and C_y
        end if
      end for
    end for
  end for

By considering Figure 5a again, we can see that the PDF of the mean coverage also represents the PDF of the clustering dimension since perfect clustering groups the multiple reads of the same strand together. In Figure 5b, we have a representation of the performance of the clustering algorithm modeled in DNAssim. The metric to evaluate the goodness of the clustering algorithm is the percentage of strands that do not have a cluster to group with. This issue may happen either because there are too many errors and the algorithm is not able to find the most similar cluster for a particular strand, or if there are few reads hampering the process. On the one hand, a high number of reads can cause this situation, so that is not desirable, but on the other hand, sufficient coverage is required to find out a cluster of the proper size to compare with. In any case, we can see that in both situations the DNAssim clustering algorithm performs very nicely since the percentage of unclustered strands is very low. After *Clustering*, a *Trace reconstruction* step is mandatory. Once we have identified different clusters, we want to choose the best candidate for each cluster before triggering the decoding operations. Several alignment and consensus algorithms exist for trace reconstruction [14,23], which are supported by the DNAssim software engine. Index and synchronization markers are then removed, and the DNA symbols are finally remapped into binary symbols.

The last step is the *Decode* operation which performs the de-randomization used in the encoding step to reduce the synthesis errors. In Figure 5c, we simulated an input to the *Outer Code decoding* step. We assessed the percentage of erasures (i.e., the percentage of nucleotides that are never read) and the percentage of erroneous nucleotides that ECC must correct as a function of the mean coverage. Straightforwardly, high mean coverage is a way to decrease both errors. In any case, the percentage values are very different in magnitude and allow speculation about which kind of errors the ECC struggles to correct when providing the stored information to the user.

3.4. Qualitative Evaluation of DNAssim

The benefit of having a DNA-based storage simulator is that it is easier to compare the different algorithms used in the encoding and decoding pipeline. Since the result of the DNA pipeline is composed of the synergy of all the algorithms, it is very important to understand how they work together and compare the results of the simulations at different steps of the decoding process. In Figure 6, we have an example of two different end-to-end simulations (i.e., experiments) using the same indexing strategy and trace reconstruction algorithm, but relying on a different outer correction code and clustering algorithm. The noise applied to the stored data is the same in both cases and the use and IDS error rate is equal to 1%. We simulated the data storage of 20 DNA strands. With the simulation parameters considered, we expect theoretically that our stored data will be reconstructed using 20 clusters and that all the failing DNA strands will be recovered at the end of the decoding process.

The first output of DNAssim is the result of the clustering algorithm used in data reconstruction. In Figure 6a, we show how many clusters the algorithm found and their size

(the number of strands composing the cluster). We can easily note that the two simulations behave differently since the first experiment found 19 clusters (one missing) and the second one 18 (two missing).

The second output is the number of errors before applying the outer code (see Figure 6b). As it can be seen, when an erasure is found by DNAssim, we have all the nucleotides in a DNA strand that are missing, and the corresponding cluster ID will show a high number of errors (ID 10 and 20 in our experiments), while in all other strands, there are subtle differences. At the end of the decoding process, DNAssim will calculate the number of recovered strands. In this simulation, even if the chosen clustering algorithm performed better for experiment #1 (i.e., a higher number of clusters was found); the combination with the other algorithms in the storage pipeline (e.g., the outer code) will perform worst since the decoding process will recover 19 strands out of 20, whereas all the strands for experiment #2 will be recovered.

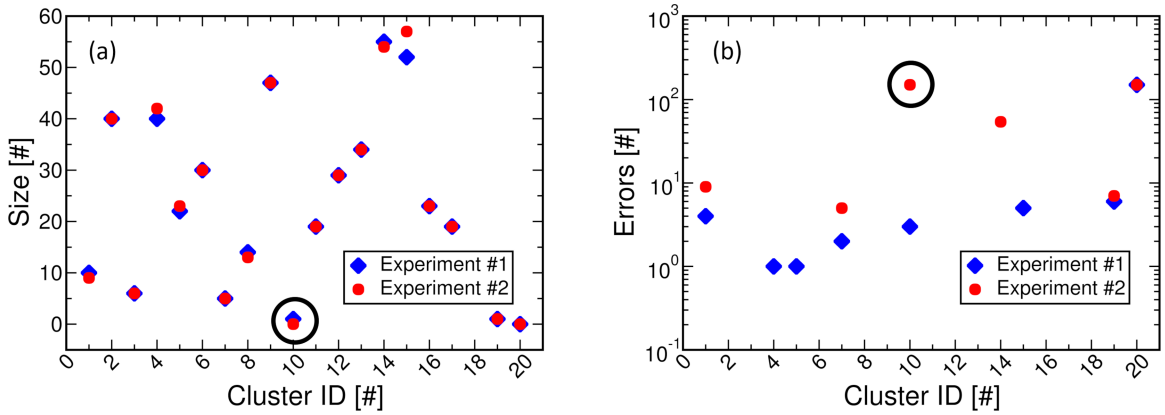

Figure 6. Two experiments performed to qualitatively assess the performance of DNAssim. In (**a**), the size of the found clusters is plotted versus the cluster ID. The black circle indicates that in Experiment #2 there are no DNA strands associated with cluster ID 10. In (**b**), we plot the errors before outer code application as a function of the cluster ID. The black circle indicates that for Experiment #2 the cluster ID 10 has been identified as missing (full DNA strand in error) and cluster ID 20 is missing in both experiments due to the noise imposed on the stored data.

4. FPGA Hardware Acceleration of the Edit Distance Computation

The identification of the edit distance as a bottleneck for the DNA storage simulation process motivated us to embrace a co-simulation approach that relies on custom hardware acceleration. In this section, we present an FPGA-based edit distance accelerator discussing its design flow from hardware to software implementation. The reasons behind the choice of an FPGA architecture for accelerating the simulations performed by DNAssim are two-fold. Firstly, the application targeted in this work (i.e., edit distance computation) is renowned as an algorithm that does not require complex computations in the floating-point domain nor fast tensor operations as normally happens for GPU-based applications. All the operations in edit distance are mainly bit-wise (bit manipulations) and are performed relying on a digital design with custom precision, which is a preferential area for FPGA designs. Secondly, FPGAs are preferred to general-purpose hardware (e.g., CPUs and GPUs) for their lower power consumption/higher energy efficiency during operation and for better handling of a large amount of data with massive dependencies. In large DNA-storage simulation campaign contexts such as the one presented in [23], these become important non-functional properties to evaluate in the final system design. Moreover, their optimization is mandatory for future scale-out simulation platforms where multiple accelerators are sought to be integrated.

4.1. Hardware Design

The FPGA card used in this work for edit distance acceleration is a Xilinx Virtex-7 VC707 evaluation kit based on the Virtex-7 XC7VX485T device. The design suite used for the development of the entire system is Vivado 2019.1.3. The accelerator architecture is presented in Figure 7. Its backbone features 32 concurrent computational blocks (CB), each one devoted to the calculation of a specific instance of the edit distance using the algorithm presented in the previous section.

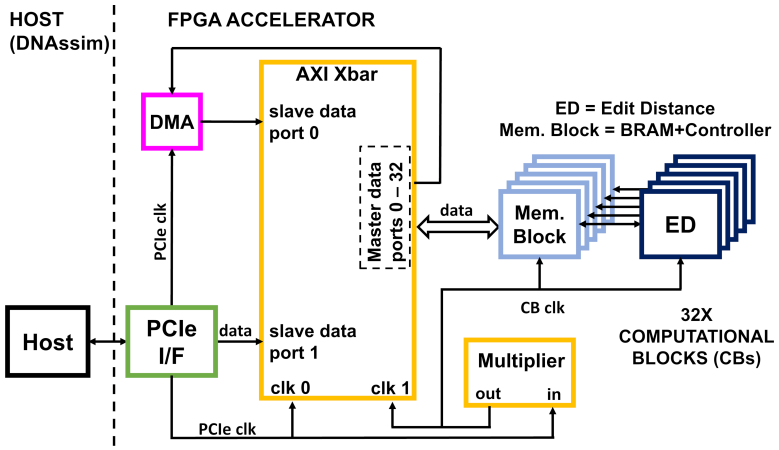

Figure 7. Block diagram of the FPGA accelerator architecture. The connection with the host system running the DNAssim software engine is highlighted as well.

4.1.1. Implementation of CBs and the Overall Framework

The implementation of the computational blocks has been performed by using Vivado High-Level Synthesis (HLS). The FPGA resources occupation for a single instance is provided in Table 1, materializing in a usage (Lookup Tables—LUT and Flip Flops—FF) per CB of lower than 3% of the FPGA area. The communication and control flow with the CBs is ensured through a set of 32 programmable 4 kB dual port block RAMs (BRAMs) that represent the main interface between the DNAssim software engine running on the host and the CBs in FPGA. It is worth noting that the actual parallelism of the backbone can be assumed in future works either as a design parameter or as a reconfigurable option to be changed before running the simulations. Indeed, a user could easily change either the number of instances or the size of the interfacing BRAMs to fit both the host and the acceleration requirements. The I/O communication path linking the host to the accelerator BRAMs is achieved through an AMBA-AXI-compliant crossbar [41] featuring 33 master data ports (one for each CB and a DMA port) and 2 slave data ports. The latter ports are connected, respectively, to a PCI Express gen2 $\times 4$ interface following the guidelines of [20] to guarantee the accelerator/host connectivity with this protocol, and to a DMA engine responsible for the accelerator internal data movements relieving the host from managing the transfers. Following this design approach, the address space covered by the BRAMs (i.e., a maximum of 128 kB for 32 CBs) can be exposed to the host as a writable/readable monolithic address region by means of a PCIe Base Address Register (BAR). Finally, to precisely control the accelerator's clock frequency, we inserted a clock multiplier block. Its role is to decouple the PCIe reference clock at 125 MHz from the one used in computational logic (170 MHz). This mesochronous time domain approach provides the maximum FPGA area occupation while keeping the frequency of the computational blocks as high as possible.

Table 1. FPGA utilization report from Vivado-HLS for a single edit distance computational block.

Name	FF	LUT
Expression	0	5915
Instance	135	118
Memory	272	96
Multiplexer	-	1849
Register	4266	-
Total	4633 (0.76%)	7978 (2.62%)
Available	607,200	303,600

4.1.2. BRAMs Design for Data Transfers

The hardware design to speed up the computation of the edit distance is based on BRAM blocks instantiated for each computational block that can store up to 4 kB of data. The width of the BRAMs has been set to 128 bits; therefore, each BRAM will have 256 addresses available. In a single BRAM instance, we choose to store up to seven couples of DNA strands with a maximum length of 254 nucleotides, which corresponds to 3556 stored bytes. This design choice is ascribed to the fact that indices of the score matrix are, in our application, ranging from 1 through 255. Only 7 bytes (strand length of 254) are needed to store up to seven results in the same BRAM block. In total, 32 BRAMs coupled with 32 CBs are implemented, which allows for calculating up to 224 results (DNA pairs). In addition to the bytes reserved for the strands and results, another 16 bytes of metadata are used—14 bytes to store each length of the strands and the other 2 bytes to control the execution of the hardware CBs. In Figure 8, we show the layout of the BRAM structure. All these considerations lead to an 87.4% occupation of BRAMs.

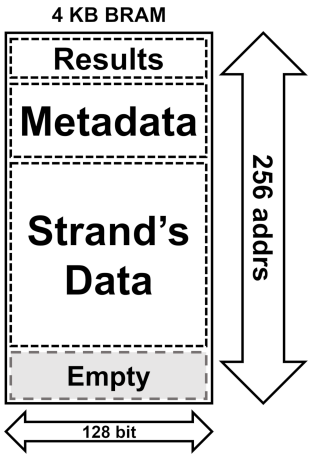

Figure 8. Addressing space layout of a single BRAM instance in the design.

4.1.3. Full Design Results

The full design with 32 CBs has been synthesized and implemented in Vivado using the default strategies provided by the development suite, resulting in the floorplan depicted in Figure 9. The estimated power consumption is 7.46 W with a Worst Negative Slack (WNS) of 136 ps. In Table 2, we show the utilization report of Vivado, indicating heavy resource usage in the design. In Figure 10, we report the performance of the implemented accelerator in terms of k (kilo) edit distance calculated per second as a function of the DNA strand length. We remember that 32 CB instances are concurrently addressed in the algorithm execution flow. The accelerator's performance is extracted using the Xilinx Integrated Logic Analyzer (ILA) IP core [42]. With DNA strand lengths between 120 and 254 (the current hardware limit), we can sustain a maximum performance of up to 700 kedit/s. It can be observed that performance decreases as the strand length increases following a power law rule with exponent -1.91. This is due to the $O(N \cdot M)$ complexity of the algorithm [29,30] where N and M are the lengths of the strands, respectively. It is worth highlighting that, in

any case, the FPGA realization of the edit distance drastically outperforms the software (C wrapped in Python) version.

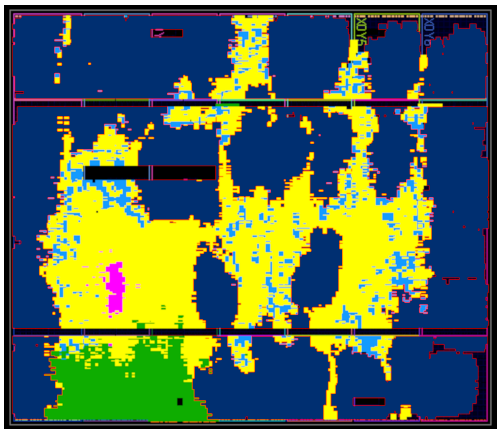

Figure 9. Floorplan of the XC7VX485T FPGA implementing a 32 CBs edit distance hardware accelerator. The colors of the hardware blocks match those indicated in Figure 7.

Table 2. Utilization report of a 32 CBs edit distance hardware accelerator implemented on the Xilinx XC7VX485T FPGA.

Site Type	Used	Available	Usage %
Slice LUTs	196,785	303,600	64.82
LUT as Logic	174,277	303,600	57.40
LUT as Memory	22,508	130,800	17.21
LUT as distrib. RAM	21,304	-	-
LUT as Shift Reg.	1204	-	-
Slice Registers	237,117	607,200	39.05
Register as FF	237,177	607,200	39.05
Register as Latch	0	607,200	0.00
F7 Muxes	5065	151,800	3.34
F8 Muxes	1404	75,900	1.85

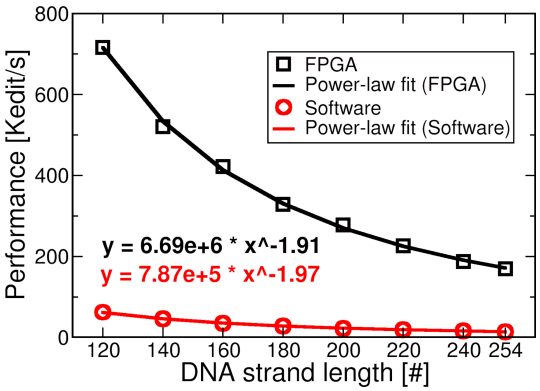

Figure 10. Performance of the FPGA-based accelerator measured in terms of calculated kedit/s as a function of the DNA strand length. A comparison with a software (Python) realization of the edit distance is provided as well.

4.2. Software Driver Design

The purpose of the software driver is to enable hardware acceleration, thus ensuring both low latency and a fast integration path with the DNAssim software engine. The software driver is written using the C programming language and must perform all the host-to-accelerator data transfers using the DMA engine implemented in the FPGA. Since the DNAssim software engine is developed in Python, the integration between the simulator

and the driver is performed using the Python/C API which acts as a bridge ensuring negligible latency. The driver is then included in a custom Python library; therefore, it can be rapidly integrated inside the simulator and the hardware acceleration of the edit distance is achieved with a simple function call. In Figure 11, we show the block diagram of the driver and how it connects the host and the FPGA design.

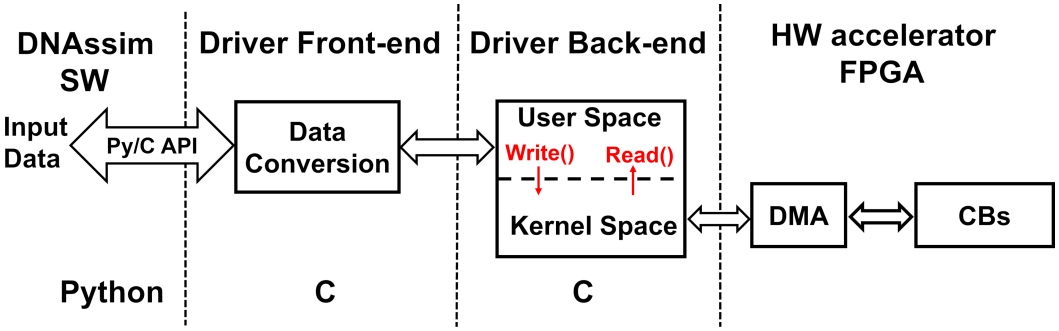

Figure 11. Block diagram of the software driver interconnecting the DNAssim software with the developed hardware accelerator.

4.2.1. Front-End

The driver's front end reshapes the DNA strands received from the software engine running on the host, appends the metadata, and writes them into a buffer. The data structure written inside the buffer is defined by the hardware accelerator. Since the number of received DNA strand pairs and their lengths can change, a zero-padding operation is included as a part of the reshaping process to match the data structure required by the hardware accelerator. The buffer is composed of 4 kB chunks representing the working memory of each CB in the FPGA (see Figure 8). Since the maximum number of CBs is 32, the maximum buffer size is 128 kB.

4.2.2. Back-End

The driver's back end performs all the I/O operations at the Operating System (OS) level. A low-latency data path is ensured by managing the transfers in the OS kernel space. Once the data buffer has been prepared by the front end, the data is copied to a pre-allocated buffer located in the kernel space (exposed to the DMA in the accelerator). After that, the DMA is programmed with the source and destination addresses, together with the amount of data to be transferred. At this point, the BRAMs associated with the CBs are loaded and the computation starts. The driver's back end then waits for the completion of the hardware accelerator computation. This state is checked with a polling approach. When the results of the hardware accelerator execution are ready, they are then transferred to the host by means of the host's CPU. Currently, no power limits on the PCI Express bus were set and for the sake of simplicity, a polling approach to check the end of data movement has been used rather than an MSI-X interrupt approach [20]. The accelerator-to-host data transfers do not use the DMA. This is because the amount of data transferred is not sufficient to benefit from the presence of DMA (the maximum results size is 224 bytes) resulting in an overhead that would defeat its use.

4.2.3. Performance

The performance achieved by the driver is measured with Python using the *time* function. It is important to underline that the measures include both write and read transactions as well as the front end latency, but do not consider the latency introduced by the hardware execution of the CBs. Indeed, it is important to assess these figures of merit in isolation. Firstly, we characterized the read/write bandwidth offered by the BRAMs on the FPGA accelerator. Measurements are performed without accounting for the execution time of the CBs and they consider only the software driver without the simulation engine triggering the transfers. The experimental results are obtained with the *gettimeofday* function

of the C language. As shown in Figure 12, both read and write bandwidths increase with the amount of transferred data (from 4 kB to 256 kB) until they become almost constant because of the saturation of the data bus. In Figure 13, the buffer of data to be transferred into the BRAMs is set to 128 kB to have enough data to compute 224 edit distances, so the host is expected to read back 224 bytes (i.e., the comparison result). It is shown that the bandwidth grows as the length of the DNA strands increases. This behavior is attributed to the overhead reduction introduced by the zero-padding operation when a buffer of a certain size (in multiples of 4 kB) is created. In Figure 14, the length of the strands is swept from 120 to 254 to match the bounds of Figure 4. Results show that the bandwidth grows until saturation as the strands couples increase, likely due to the data bus saturation experienced in Figure 12.

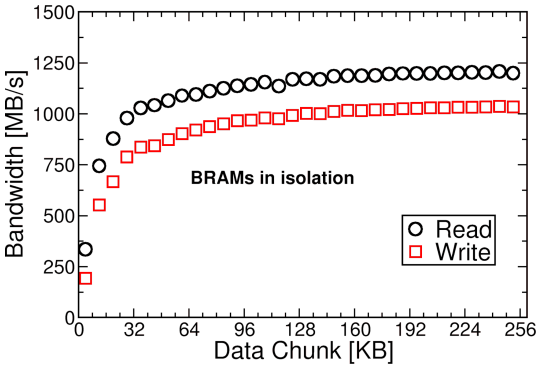

Figure 12. Characterization of the BRAMs read and write bandwidth in isolation from the CBs.

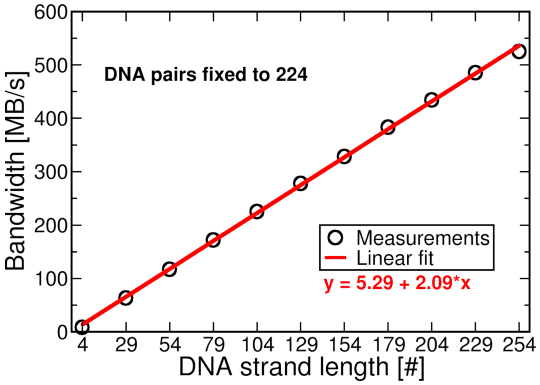

Figure 13. Driver's bandwidth (front end and back end) in isolation as a function of the DNA strand length.

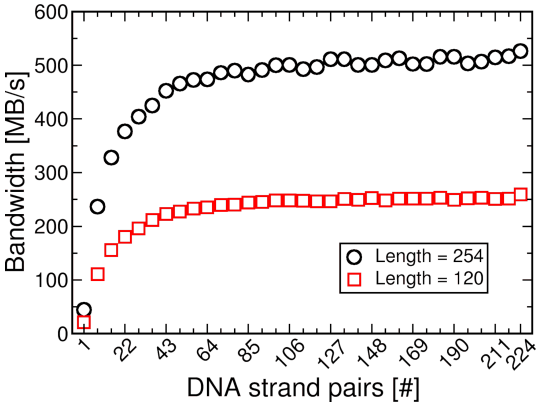

Figure 14. Driver's bandwidth (front end and back end) in isolation as a function of the number of DNA strand pairs.

5. Co-Simulation Experiments and Results

To assess the performance benefits of our proposed hardware/software co-simulation approach for DNA-based storage, we put the DNAssim framework and the FPGA accelerator under test on a time-consuming operation such as clustering. A massive amount of edit distance calculations are expected in this context, as discussed in the previous section of this work. We measured, through the *time* function of a Python library, the clustering latency obtained with different simulation parameters. The aim is to thoroughly explore the areas with a performance advantage with respect to a pure software approach.

The test rig considered for all the experiments is shown in Figure 15. It is equipped with an Asus Prime Z590-P motherboard which mounts an Intel Core i5-10600K CPU with a 4.1 GHz clock frequency. The memory is a 16 GB (2 × 8 GB DIMMs) DDR4 Hyperx Predator with a frequency equal to 2666 MHz. The storage is a 500 GB WD BLUE M.2 2280 connected to a PCIe gen3 ×4 interface. The Xilinx VC707 evaluation kit is mounted on a PCIe slot available on an external PCIe gen3 ×16 PLX switch and connected to the PC motherboard with extender cables. The OS installed is Linux Ubuntu 20.04.

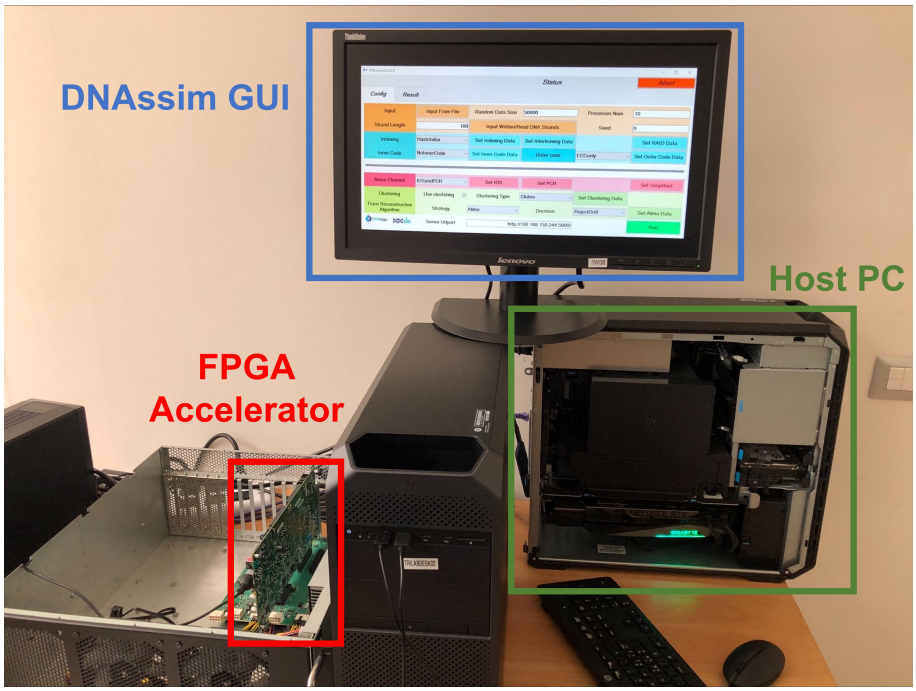

Figure 15. A photograph of the test rig used to assess the performance of the DNAssim framework. The Graphical User Interface (GUI) of the software engine and the FPGA-based hardware accelerator attached to the host PC are highlighted.

In Figure 16, we report the simulation latency of the clustering operation during the decoding phase in DNA-based data storage. We considered the mean coverage parameter as 10, 20, and 40 and we varied the DNA strand length with values 120, 160, and 200. To increase the space of exploration, we also considered different amounts of strings that are input to the clustering operation, namely 10^5, 10^6, and 10^7. From the figure, it is possible to straightforwardly observe that the higher the amount of input strings, the longer the clustering simulation (from hundredths of seconds up to tens of minutes). Interestingly, we note that our co-simulation approach can accelerate the clustering operation up to a factor of 5.5 for low mean coverage and short DNA strands. In any case, the number of acceleration scales with the mean coverage and the number of strings in input support the use of our approach to speed up DNA-based data storage simulations. To further qualitatively support the benefit of our proposed approach, we project the obtained speed-up results to the clustering of real DNA data storage use cases by relying on the datasets provided in [43] and by approximating the number of reads and the average strand length

to our test conditions of Figure 16. As shown in Table 3, a maximum speed-up of 4.2× can be achieved in the simulation of a music file stored in the DNA using a co-simulation approach with the developed FPGA accelerator.

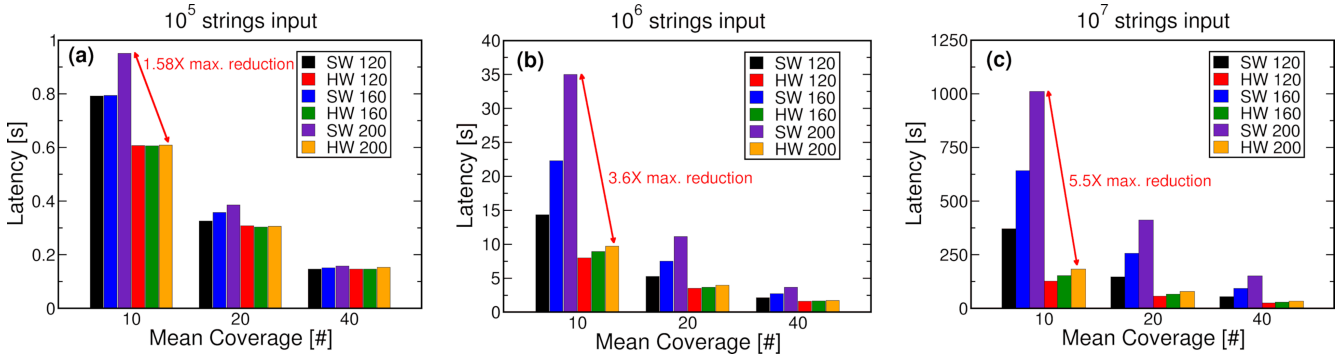

Figure 16. Clustering simulation latency benchmark between a pure software approach (SW) and our acceleration framework (HW). The simulation parameters varied in the analysis are the mean coverage and the DNA strand length. We consider different strings number in input to clustering equal to 10^5 (**a**), 10^6 (**b**), and 10^7 (**c**), respectively.

Table 3. Range of speed-ups achievable in real DNA stored datasets (data traces from Organick [43]) using our co-simulation approach with respect to a pure software execution of DNAssim. The number of reads and the average strand length are approximated using the closest value in excess used in our clustering latency experiments.

Dataset	# Reads (Closest)	Avg. Strand Length (Closest)	Description	Speed-Up
3.1M	3,103,511 (10^6)	150 (160)	Movie file	2.49×
13.2M	13,256,431 (10^7)	150 (160)	Music file	4.2×
12M	11,973,538 (10^7)	110 (120)	Text file	2.91×

6. Conclusions

In this work, we have presented the integration of an FPGA hardware accelerator dedicated to editing distance calculation with a DNA-based data storage simulation environment (DNAssim) devoted to the study of the information encoding/decoding pipeline. The accelerator has been developed on a Xilinx VC707 evaluation kit with a Virtex-7 XC7VX485T FPGA and features a PCIe gen2 ×4 driver connected with DNAssim. The design has been implemented at 170 MHz exhibiting a computation performance of 700 kedit/s (11 times the performance of a pure software edit distance implementation) with a power consumption of 7.46 W. The combination of the FPGA accelerator and DNAssim has been validated in the simulations of the clustering operation process during the decoding phase of the stored data on DNA. Massive simulation time reductions of up to 5.5 times are reported using the proposed co-simulation approach and a projection on real datasets forecasts a time reduction of 4.2 times in the simulation of the storage of a music file on the DNA.

Future works will be oriented to the extension of the DNAssim platform with heterogeneous integration of multiple accelerators and in the direction of large scale-out DNA storage simulations.

Author Contributions: The individual contributions to this paper are the following: conceptualization, A.M., C.Z. and R.M.; methodology, A.M., C.Z. and R.M.; software, A.M., T.C., N.B. and L.Z.; validation, T.C., N.B. and L.Z.; electrical measurements and hardware implementation, T.C., N.B. and L.Z.; investigation, A.M., C.Z. and R.M.; data curation, C.Z. and R.M.; writing—original draft preparation, A.M. and C.Z.; writing—review and editing, C.Z. and R.M.; visualization, C.Z.; supervision, R.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank G. Lanzoni for their help in the design of the FPGA accelerator and for fruitful discussion.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DNA	DeoxyriboNucleic Acid
PCR	Polymerase Chain Reaction
FPGA	Field Programmable Gate Array
IDS	Inserted, Deleted, and transposed/Substituted
ECC	Error Correction Codes
BRAM	Block Random Access Memory

References

1. Rydning, J.; Reinsel, D. *Worldwide Global StorageSphere Forecast, 2021–2025: To Save or Not to Save Data, That Is the Question*; Technical Report IDC Doc #US47509621; IDC Corp.: Needham, MA, USA, 2021.
2. Wieder, P.; Butler, J.M.; Theilmann, W.; Yahyapour, R. *Service Level Agreements for Cloud Computing*; Springer: New York, NY, USA, 2014. [CrossRef]
3. DNA Data Storage Alliance. *Preserving Our Digital Legacy: An Introduction to DNA Data Storage*; Technical Report; 2021. Available online: https://dnastoragealliance.org/dev/wp-content/uploads/2021/06/DNA-Data-Storage-Alliance-An-Introduction-to-DNA-Data-Storage.pdf (accessed on 6 June 2023)
4. Alberts, B.; Bray, D.; Lewis, J.; Raff, M.; Roberts, K.; Watson, J. *Molecular Biology of the Cell*, 4th ed.; Garland: New York, NY, USA, 2002.
5. Erlich, Y.; Zielinski, D. DNA Fountain enables a robust and efficient storage architecture. *Science* **2017**, *355*, 950–954. [CrossRef] [PubMed]
6. Grass, R.; Heckel, R.; Puddu, M.; Paunescu, D.; Stark, W. Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes. *Angew. Chem. Int. Ed.* **2015**, *54*, 2552. [CrossRef] [PubMed]
7. DNA Storage. 2015. Available online: https://www.microsoft.com/en-us/research/project/dna-storage/ (accessed on 15 April 2023).
8. Budel, S. Next Generation Sequencing (NGS) Market Assessment Trends (2018–2024); Technical Report; DeciBio: Los Angeles, CA, USA, 2021.
9. Brown, K. A $100 Genome within Reach, Illumina CEO Asks If World Is Ready. 2019. Available online: https://www.bloomberg.com/news/articles/2019-02-27/a-100-genome-within-reach-illumina-ceo-asks-if-world-is-ready (accessed on 15 April 2023).
10. Genscript. Gene Synthesis & DNA Synthesis Service. 2023. Available online: https://www.genscript.com/gene_synthesis.html?src=google&gclid=Cj0KCQjwyLGjBhDKARIsAFRNgW_Y6C7bL0pr-U_MZA_2tmShoNPCZWmjEZuLPCm4OjBff-LARSzPE3oaAu3BEALw_wcB (accessed on 24 April 2023).
11. Saiki, R.; Gelfand, D.; Stoffel, S.; Scharf, S.; Higuchi, R.; Horn, G.; Mullis, K.; Erlich, H. Primer-directed enzymatic amplification of DNA with a thermostable DNA polymerase. *Science* **1988**, *239*, 487–491. [CrossRef]
12. Chandak, S.; Tatwawadi, K.; Lau, B.; Mardia, J.; Kubit, M.; Neu, J.; Griffin, P.; Wootters, M.; Weissman, T.; Ji, H. Improved Read/Write Cost Tradeoff in DNA-Based Data Storage Using LDPC Codes. In Proceedings of the 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Monticello, IL, USA, 24–27 September 2019; pp. 147–156. [CrossRef]
13. Mitzenmacher, M. A survey of results for deletion channels and related synchronization channels. *Probab. Surv.* **2009**, *6*, 1–33. [CrossRef]
14. Church, G.M.; Gao, Y.; Kosuri, S. Next-generation digital information storage in DNA. *Science* **2012**, *337*, 1628. [CrossRef]
15. Goldman, N.; Bertone, P.; Chen, S.; Dessimoz, C.; LeProust, E.M.; Sipos, B.; Birney, E. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature* **2013**, *494*, 77–80. [CrossRef] [PubMed]
16. Blawat, M.; Gaedke, K.; Hütter, I.; Chen, X.M.; Turczyk, B.; Inverso, S.; Pruitt, B.W.; Church, G.M. Forward Error Correction for DNA Data Storage. *Procedia Comput. Sci.* **2016**, *80*, 1011–1022. [CrossRef]
17. Tabatabaei Yazdi, S.M.H.; Gabrys, R.; Milenkovic, O. Portable and Error-Free DNA-Based Data Storage. *Sci. Rep.* **2017**, *7*, 5011. [CrossRef]
18. Organick, L.; Ang, S.; Chen, Y.J.; Lopez, R.; Yekhanin, S.; Makarychev, K.; Racz, M.; Kamath, G.; Gopalan, P.; Nguyen, B.; et al. Random access in large-scale DNA data storage. *Nat. Biotechnol.* **2018**, *36*, 242–248. [CrossRef]

19. Heckel, R.; Mikutis, G.; Grass, R.N. A Characterization of the DNA Data Storage Channel. *Sci. Rep.* **2019**, *9*, 9663. [CrossRef]
20. AXI Memory Mapped to PCI Express (PCIe) Gen2 v2.9. 2021. Available online: https://docs.xilinx.com/v/u/en-US/pg055-axi-bridge-pcie/ (accessed on 18 April 2023).
21. Marelli, A.; Chiozzi, T.; Zuolo, L.; Battistini, N.; Lanzoni, G.; Olivo, P.; Zambelli, C.; Micheloni, R. DNAssim: A Full System Simulator for DNA Storage. In Proceedings of the Flash Memory Summit, Santa Clara, CA, USA, 8–10 August 2022.
22. Marelli, A.; Chiozzi, T.; Zuolo, L.; Battistini, N.; Olivo, P.; Zambelli, C.; Micheloni, R. DNAssim: A Full System Simulator for DNA Storage. In Proceedings of the Storage Developer Conference, Fremont, CA, USA, 12–15 September 2022.
23. Rashtchian, C.; Makarychev, K.; Racz, M.; Ang, S.; Jevdjic, D.; Yekhanin, S.; Ceze, L.; Strauss, K. Clustering Billions of Reads for DNA Data Storage. In *Proceedings of the Advances in Neural Information Processing Systems 30*; MIT Press: Cambridge, MA, USA, 2017.
24. Whitwam, R. Microsoft Automates DNA-Based Data Storage. 2019. Available online: https://www.extremetech.com/extreme/288240-microsoft-automates-dna-based-data-storage (accessed on 15 April 2023).
25. Lassmann, T.; Frings, O.; Sonnhammer, E. Kalign2: High-performance Multiple Alignment of Protein and Nucleotide Sequences Allowing External Features. *Nucleic Acids Res.* **2009**, *37*, 858–865. [CrossRef] [PubMed]
26. Srinivasavaradhan, S.R.; Gopi, S.; Pfister, H.D.; Yekhanin, S. Trellis BMA: Coded Trace Reconstruction on IDS Channels for DNA Storage. In Proceedings of the 2021 IEEE International Symposium on Information Theory (ISIT), Melbourne, Australia, 12–20 July 2021; pp. 2453–2458. [CrossRef]
27. Zuolo, L.; Zambelli, C.; Marelli, A.; Micheloni, R.; Olivo, P. LDPC Soft Decoding with Improved Performance in 1X-2X MLC and TLC NAND Flash-Based Solid State Drives. *IEEE Trans. Emerg. Top. Comput.* **2019**, *7*, 507–515. [CrossRef]
28. Zuolo, L.; Zambelli, C.; Micheloni, R.; Indaco, M.; Carlo, S.D.; Prinetto, P.; Bertozzi, D.; Olivo, P. SSDExplorer: A Virtual Platform for Performance/Reliability-Oriented Fine-Grained Design Space Exploration of Solid State Drives. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2015**, *34*, 1627–1638. [CrossRef]
29. Caffarena, G.; Pedreira, C.; Carreras, C.; Bojanic, S.; Nieto-Taladriz, O. FPGA Acceleration for DNA sequence alignment. *J. Circuits Syst. Comput.* **2007**, *16*, 245–266. [CrossRef]
30. Kent, K.; Proudfoot, R.; Zhao, Y. Parameter-Specific FPGA Implementation of Edit-Distance Calculation. In Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping (RSP'06), Chania, Greece, 14–16 June 2006; pp. 209–215. [CrossRef]
31. Dydel, S.; Bała, P. Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices. In Proceedings of the Field Programmable Logic and Application; Becker, J., Platzner, M., Vernalde, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 23–32.
32. Castells-Rufas, D.; Marco-Sola, S.; Moure, J.C.; Aguado, Q.; Espinosa, A. FPGA Acceleration of Pre-Alignment Filters for Short Read Mapping with HLS. *IEEE Access* **2022**, *10*, 22079–22100. [CrossRef]
33. Marchisio, A.; Teodonio, F.; Rizzi, A.; Shafique, M. ISMatch: A real-time hardware accelerator for inexact string matching of DNA sequences on FPGA. *Microprocess. Microsystems* **2023**, *97*, 104763. [CrossRef]
34. Cai, K.; Chee, Y.M.; Gabrys, R.; Kiah, H.M.; Nguyen, T.T. Correcting a Single Indel/Edit for DNA-Based Data Storage: Linear-Time Encoders and Order-Optimality. *IEEE Trans. Inf. Theory* **2021**, *67*, 3438–3451. [CrossRef]
35. Leung, K.; Welch, L. Erasure decoding in burst-error channels. *IEEE Trans. Inf. Theory* **1981**, *27*, 160–167. [CrossRef]
36. Skiena, S.S., Hashing and Randomized Algorithms. In *The Algorithm Design Manual*; Springer: Cham, Switzerlands, 2020; pp. 171–195. [CrossRef]
37. Shomorony, I.; Heckel, R. DNA-Based Storage: Models and Fundamental Limits. *IEEE Trans. Inf. Theory* **2021**, *67*, 3675–3689. [CrossRef]
38. Mao, W.; Diggavi, S.N.; Kannan, S. Models and Information-Theoretic Bounds for Nanopore Sequencing. *IEEE Trans. Inf. Theory* **2018**, *64*, 3216–3236. [CrossRef]
39. Berger, B.; Waterman, M.S.; Yu, Y.W. Levenshtein Distance, Sequence Comparison and Biological Database Search. *IEEE Trans. Inf. Theory* **2021**, *67*, 3287–3294. [CrossRef] [PubMed]
40. Navarro, G. A Guided Tour to Approximate String Matching. *ACM Comput. Surv.* **2001**, *33*, 31–88. [CrossRef]
41. AMBA AXI4 protocol. 2019. Available online: https://developer.arm.com/products/architecture/system-architectures/amba/amba-4 (accessed on 15 April 2023).
42. Xilinx Integrated Logic Analyzer (ILA) v2.0 IP-Core. 2012. Available online: https://docs.xilinx.com/v/u/en-US/ds875-ila (accessed on 15 April 2023).
43. Organick, L.; Ang, S.D.; Chen, Y.J.; Lopez, R.; Yekhanin, S.; Makarychev, K.; Racz, M.Z.; Kamath, G.; Gopalan, P.; Nguyen, B.; et al. Scaling up DNA data storage and random access retrieval. *bioRxiv* **2017**. [CrossRef]