Accelerating Quantum Error Correction on FPGA: the Fusion Blossom approach

Niccolò Brembilla

June 30, 2024

POLITECNICO
MILANO 1863

POLITECNICO MILANO 1863
NECST laboratory

Abstract

Quantum Computing (QC) represents a novel computing paradigm with the ability to revolutionize the computer science domain, as it paves the way for solving problems that the current generation of classical devices is not able to deal with. Nevertheless, qubits in these systems are not perfect and suffer from noise. Therefore, finding ways to correct errors introduced in the computation becomes of paramount importance to enable QC to scale. In this context, Quantum Error Correction (QEC) represents a valuable approach but is characterized by strict limitations in latency. To satisfy these constraints, hardware acceleration on heterogeneous architectures, such as FPGAs, proved to be an effective strategy. In this context, this project consists of the deployment of an FPGA-based solution to decode quantum errors leveraging the Fusion Blossom algorithm. This work proposes an FPGA-based approach for the acceleration of the MWPM approach used in the Fusion Blossom decoder. This solution is the first step towards the full implementation of the Fusion Blossom on FPGA. By using a host code coupled with an AMD-Xilinx Alveo U55C can provide a functional approximated up-to-distance 9 decoder that scales almost linearly on sparse graphs. This solution has shown great potential for implementing accelerated algorithms for higher-distance and faster decoders.

1 Introduction

Quantum Computing holds great promise for revolutionizing various applications such as machine learning and cybersecurity. However, the challenge of high error rates due to environmental noise and quantum decoherence must be addressed.

Quantum Error Correction relies on periodically reading physical qubits through a syndrome extraction circuit (based on stabilizers) while the system runs. The syndrome is then delivered as a decoding graph to external devices that compute the most probable error and provide the correction back to the quantum system.

This work pursues an FPGA-based approach for the Fusion Blossom algorithm acceleration to reach real-time correction speed $t < 1\mu s$, implementing a simplified version of it on a maximum distance of 9.

In particular, the implementation shows an almost linear time scaling on parse graphs and a better scaling than the software version on higher error syndromes.

To validate the overall performance the solution is tested on AMD-Xilinx Alveo U55C.

In Section 2 and Section 3, we provide the required background to better understand the solution. In Section 4, we describe the algorithm and present the overall architecture that we designed; finally, in Section 5, we provide our experimental results.

2 Background

2.1 QEC

Quantum computers hold immense potential for solving problems intractable for classical computers. However, their advantage is based on the ability to manipulate and maintain the delicate quantum states susceptible to errors. Quantum Error Correction (QEC) is crucial in mitigating these errors and ensuring reliable quantum computations.

Firstly, we need to define how we will use the data structure to represent the decoding codes, called the *model graph*. This code consists of data qubits and a stabilizer that allows the observation of errors in the data qubits. The model graph can be represented as $G_M = (V_M, E_M)$. A vertex $v \in V_M$ is the measurement result of the stabilizer, and the weighted edge $e \in E_M$ corresponds to an independent error source that connects two vertices. If an edge connects only one vertex we add a *virtual vertex* to create a two-dimensional graph.

Another important part of decoding is the Syndrome of the decoding graph, which is the outcome of the measurement of the stabilizers. A non-zero syndrome identifies a deviation from the ideal state defined by the stabilizers in the decoding graph. To solve this problem we use a Most-Likely Error (MLE) decoder to find an error pattern that generates the Syndrome. The solution is the subset of edges $\epsilon \in E_M$ that maximize the probability of errors in the decoding graph, so our objective is to minimize $W(\epsilon) = \sum_{e \in \epsilon} w_e$. More information on how this has been developed will be found in Section 4.

2.2 Blossom

The blossom algorithm is an effective technique for finding a minimum-weight perfect matching in a bigraph. The core idea of the Blossom algorithm is to identify and collapse redundant edges in the matching graph (the syndrome graph in a QEC problem), vastly simplifying the graph and, consequently, making the matching search more efficient. More will be explained in Section 4.

While older off-the-shelf MPWM libraries, such as Kolmogorov's blossom V library [4] and the Lemon library by Dezs″o et al [2], computes the distances between the syndrome error nodes building the syndrome graph, more recent works such as the Parity Blossom and the Sparse Blossom skip entirely this part. As better described in [7] and [3], they formulate the MWPM problem as an integer linear programming (ILP) problem solving the *primal* formulation in the decoding graph and using the Blossom Algorithm to exploit the *dual* formulation of the same problem.

3 State of the Art

3.1 QEC on FPGA

As described in detail in [1], the existent solutions differ from each other on the approach, noise model, physical error rate, and most importantly, code distance. The majority of the solutions use a maximum code distance d = [3:9] and a physical error rate p=0.001, and only one of them [6] utilizes the MWPM, which is one of the main focuses of our implementation.

In recent years, many possible solutions have arisen, such as [5]: the parallelization of the Union Find on a Xilinx VCU129 FPGA. This should be a starting point to shift the focus on the use of the potential of FPGAs, in fact the solution presented in [5] is scalable both in distance and physical error rate, being faster by an order of magnitude than both [7] and [3], that are the bases of this implementation.

As this implementation, also [6] tries implementing an MWPM algorithm on an FPGA having great results on syndromes up to a distance of 7 ($decodingtime < 1\mu s$), but their solution is not able to scale further due to the exponentially increasing number of resource usage.

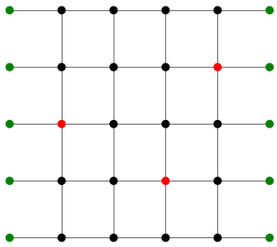
4 Implementation

4.1 Base Structures

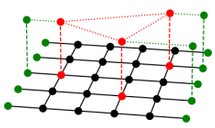
There are 4 main data structures:

- the decoding graph (Figure 1a): the two-dimensional graph in which the expansion and retraction of the error vertices are computed. Every vertex can have a defective parent (an error node has expanded over the node).

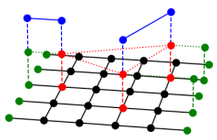
- the syndrome graph (Figure 1b): our implementation is constructed in the defective nodes. Instead of creating it directly with all possible connections between error nodes, using the Blossom algorithm, the maximum number of connections per node is two (a node can't have two weak connections or two strong connections) and they are created only if needed using the method described in Section 4.2. If a third node tries to connect to a node the connection is put



(a) Decoding graph



(b) All possible syndrome connections



(c) Pairs solution in the syndrome graph

Figure 1: Example of a decoding graph, syndrome graph, and solution: the red dots represent the error nodes, and in green are the virtual nodes. The syndrome graph edge weight is the minimum weight between the 2 nodes in the decoding graph. The pairs are the MPWM solution to the problem.

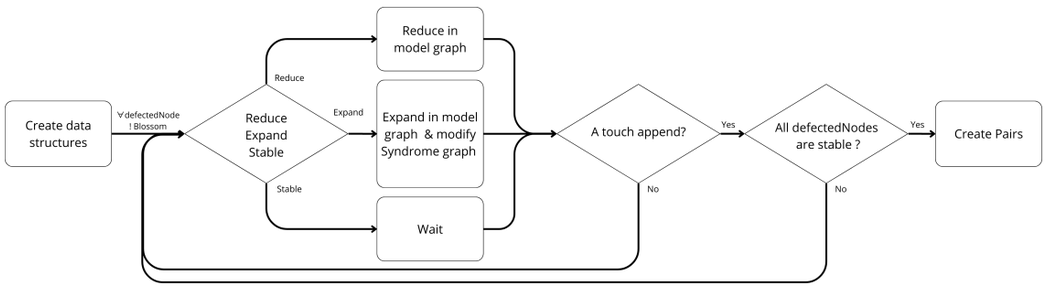


Figure 2: System workflow

in the possible connection (used if one of the other is cut). The structure also maintains the list of nodes in the decoding graph to expand or retract.

- the blossom nodes: when an odd number of defective nodes are connected in a loop, they are collapsed into a new defective node that represents the blossom node, and all information, such as the nodes that are a part of the blossom, are maintained in this data structure.

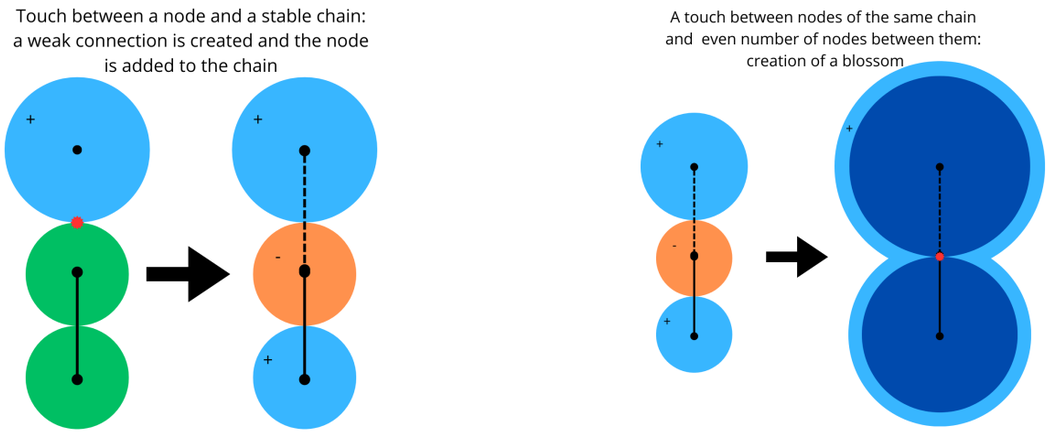
- the pairs (Figure 1c): identified in the Syndrome graph by the defective nodes connected with a strong connection.

4.2 Expansion & Retraction

In the next paragraph, the term *chain* will be used to describe a union of defective nodes connected (in the syndrome graph).
As can be seen in Figure 2 the Expansion and retraction phase is the main body of the algorithm, first there is the retraction of the nodes, and after the expansion part of the algorithm happens; if a node is part of an even dimension chain (stable chain) is nor expanded nor retracted.

At every cycle, if a defective node needs to be retracted (node in an even position in an odd length chain Figure 3b) every decoding graph node that is in the list of the defective node retracts of half the edge weight. In the end, all the possible connections (the ones that are not already constructed in the syndrome graph) are deleted. The retraction part of the algorithm for a single defective node can be fully pipelined, see Figure 5 because the various nodes' retractions don't interfere with each other.

In every other case, the defective node will be expanded. During the expansion, each node in the decoding graph will "gain" half of the weight of the near edges. If an edge weight becomes zero, the node in the decoding graph becomes part of the defective node or a touch between nodes happens:



(a) Expansion

(b) Retraction and Blossom

Figure 3: Example of expansion (Figure 3a) and retraction (Figure 3b) in the decoding graph and Blossom creation viewed from the decoding graph. For clarity, the background decoding graph has been omitted.

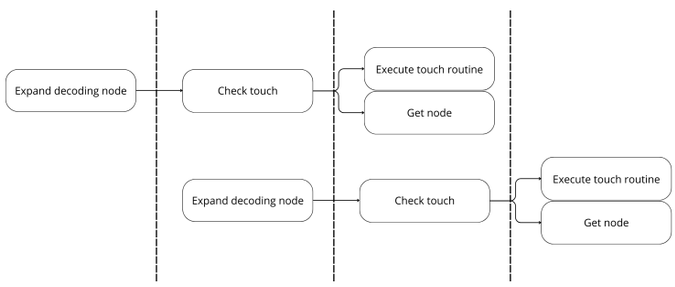


Figure 4: Expansion Pipeline

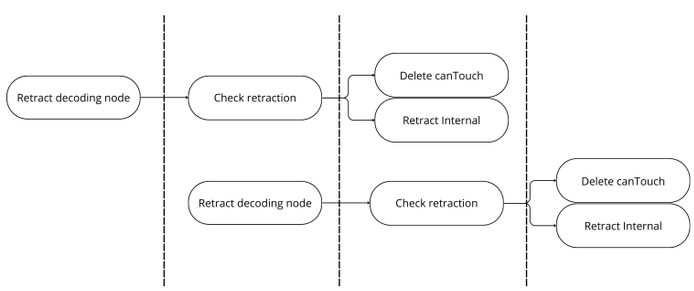


Figure 5: Retraction Pipeline

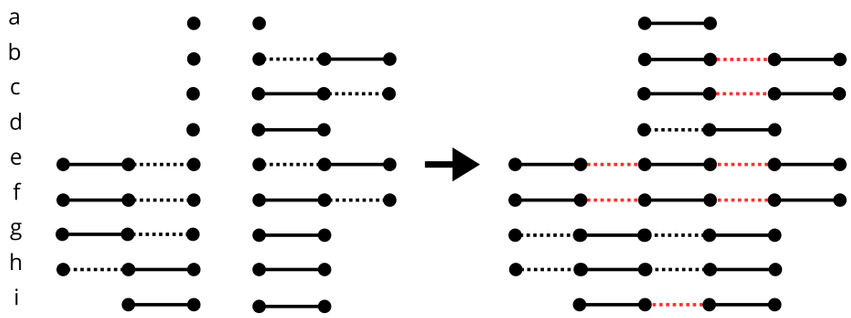


Figure 6: Representation of all possible touches in the Syndrome Graph. The dotted lines are Weak connections. Solid lines are Strong connections. Red dotted lines will be cut and become possible connections at the end of the process. The cases are numbered as per Table 1.

- If the two nodes are the same or already in touch, nothing happens.
- If the connection will create an odd loop a **Blossom is created**
- Else, as shown in Table 1 and visually explained in Figure 6, a define type of connection is created

The expansions of the decoding nodes of the same defective node can be pipelined, see Figure 4, because even though they expand on the same node the final result is the same, but to be able to have a deterministic algorithm the touches must be in order.

Case	**Node 1**		**Node 2**		**Solutions**
	No.	**Type**	**No.**	**Type**	
a	0		0		Add Strong connection
b	0		1	W(S)	Add Strong connection
c	0		1	S(W)	Change chain 2 and add Strong connection
d	0		1	S(S)	Add Weak connection
e	1	W(S)	1	W(S)	Add Strong connection
f	1	W(S)	1	S(W)	Change chain 2 and add Strong connection
g	1	W(S)	1	S(S)	Change chain 1 and add Weak connection
h	1	S(W)	1	S(S)	Add Weak connection
i	1	S(S)	1	S(S)	Add possible connection
j	-		2		Add possible connection

Table 1: Node Connections and Solutions: In the table the term change means to change all types of connections in the chain, W means weak connection, and S means strong connection. The symbol in the parenthesis defines the connection type of the last node on the other side of the chain.

(a) Not stable (b) Stable

Figure 7: In (a) the node that touches the virtual one is connected to two chains of odd length so it continues to expand/reduce. In (b) the node touching the virtual one is connected to 2 chains of even length, so all the chains are stable and the final connections are created (in red plus the connection with the virtual node)

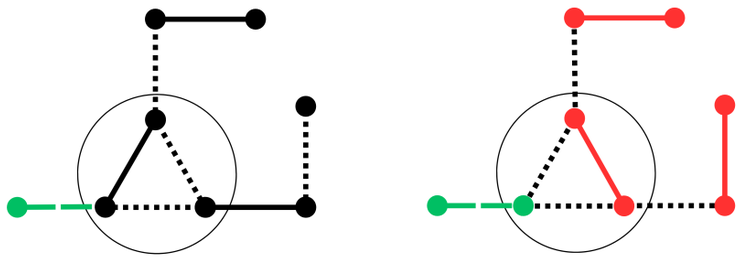


Figure 8: Example of a solution, in green the connection to a virtual node, the dotted lines are the weak connections, the full lines are the strong connections. The solution is defined by the red lines, plus the connection to the blossom

4.3 Round Check

As it can be seen in Figure 2, this phase is done after every round of expansion and retraction only if a touch happens in the syndrome graph.

In this phase, every top-level defective node or blossom is checked and only if one of these conditions is true is the chain in which the node is considered stable:

- in the chain, there are an even number of nodes.
- the chain is connected correctly to a virtual node (both branches of the chain starting from the node that is connected to the virtual node are of even dimension), see Fig...

If a chain is considered stable, its nodes will not be retracted or expanded in the next round. Otherwise, the odd position nodes in the chain will be expanded and the even ones will be retracted.

If a Blossom must be retracted in the next round and its dimension is 0, the Blossom must be deleted and the defective nodes inside of it must be divided accordingly (2 in a stable chain and one in the main chain or all 3 in the main chain).

As it can be seen in Figure 2 if all top-level nodes are stable we continue the construction of the pairs, or else the cycle continues.

Thanks to the "division" in chains the computation of the different chains can be parallelized because the algorithm online checks the length of them and/or if they are connected to virtual nodes.

4.4 Construction of the pairs

In the last part of the algorithm, the algorithm creates the pairs based on the strong connections in the Syndrome graph.

If there is a Blossom, a pair will be created between the external node that has a strong connection with the Blossom and the internal node that is connected to it and the remaining (two) internal defective nodes will be paired to each other.

In this part the algorithm can be parallelized similarly to the Round Check part, but also, as shown in Figure 8, the chain connected to a virtual node is divided into 3

Figure 9: Acceleration card AMD-Xilinx Alveo U55C

parts, that can be computed separately and the virtual node become an actual node of the solution.

4.5 Future Implementations

Due to the complexity of the algorithm, some parts of it will be implemented in future versions (these are part of the main algorithm, but not implemented on FPGA):

- The maximum dimension of the decoding graph is $d \leq 9$.
- The physical error rate is $p \leq 0.01$ (0.05 can be used but due to the restrictions it can create some problems).
- The deletion part of the algorithm is not implemented (the probability of deletion is low if an error rate $p \leq 0.01$ is used in a graph of $d \leq 9$).
- A blossom can't contain other blossoms.
- Different weight of the edges w_e.
- The solution is not the set of edges to connect but the set of pairs of vertices $(v1, v2)$ where one of the two must be an error node in the decoding graph.

5 Results

5.1 Experimental setup

Our setup consisted of a system with the AMD-Xilinx Alveo U55C High Performance Compute Card [21], shown in Figure 9, with 1,304K Look-Up Tables (LUTs), 2,607K registers, 9,024 DSP slices, and 16 GB of HBM Memory. This device has been chosen due to its adherence to the requirements for quantum error correction: it allows for high performance, a noticeable amount of available resources, and support for HBM. For the synthesis, we used Vitis HLS and Vitis version 2023.1.

5.2 Performance & Resource Usage

The Algorithm uses BRAMs (10%) to store all the information of both the decoding graph (defective node parent, near edges, position) and the syndrome graph (touches, node of the decoding graph, blossom father). The algorithm uses a good amount of LUT and DSPs due to its complexity, but most of the operations are comparisons and simple mathematical operations. The overall frequency achieved in the Synthesis is 150MHZ. In Table 2, we present the amount and the percentage of used resources for AMD-Xilinx Alveo U55C.

Table 2: Resources usage

Frequency	LUT	Register	BRAM	DSPs
150MHZ	14.6% (190320)	9.11% (237585)	10.3% (207)	7.2% (652)

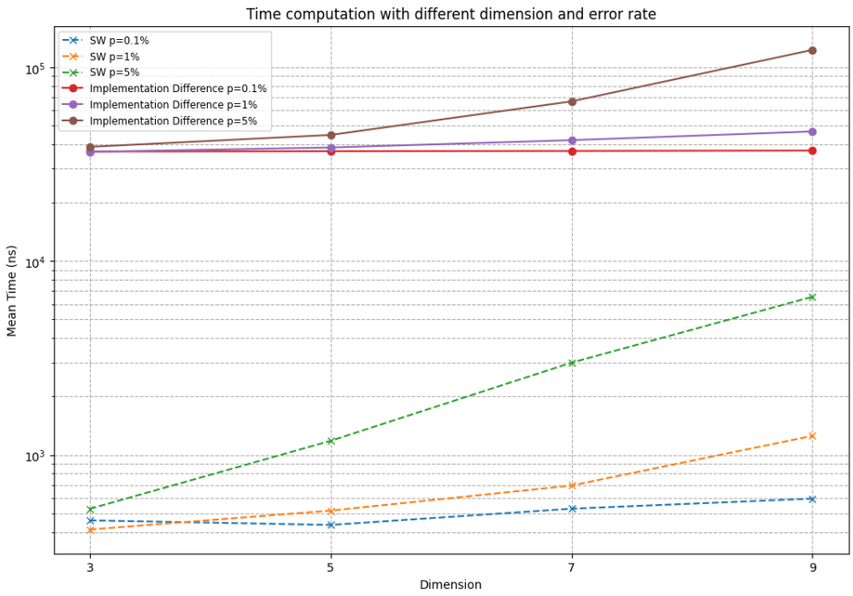


Figure 10: Comparison between the execution time for the hardware accelerator on AMD-Xilinx Alveo U55C and the SW version of the algorithm

5.3 Decoding Time & SW Comparison

As we can see in Figure 10 the accelerated version of the algorithm is slower than the software version. This is because the time computation for the accelerated version is taken from the host and not directly calculated in the kernel code; this means that the total time is also considered the transfer time of the data from the external HBM memory to FPGA's internal BRAM (this can be more than 70% of the time in the easier cases). We can also see that for the cases in which the decoding graph can be considered as a sparse graph (0.1% and 1%), the solution's execution time scales almost linearly. Concerning the SW the accelerated solution at 5% scales at a lower pace; subtracting a possible data transfer timing of $\sim 38\mu s$ the time complexity approximately appears to be roughly $O(d^3)$.

6 Conclusions

This report shows the huge potential and the reasoning behind Quantum Error Correction to resolve Quantum Computing problems and the strategies employed for the realization of QEC architectures with real-time performance. Surface codes require decoding algorithms to extract from the syndromes the most probable error patterns and compute a correction to provide to the quantum computers. MWPM solvers are used for their minimum and deterministic solution for correction. Among these algorithms, the Fusion Blossom is very promising, since in most cases can meet real-time performance constraints due to its parallel realization. This report provides the first tentative architecture to accelerate a simplified solution using an FPGA. A system that can directly receive the syndrome and provide a correct solution. This is the first step towards a full implementation of the algorithm to drastically reduce the execution time, a fundamental requirement to meet the $1\mu s$ constraint per round of correction, needed to perform real-time error correction. Additionally, multiple speed-ups are possible with further utilization of the pipeline and a wiser use of the FPGA resources; for these reasons, this is the right approach to achieve real-time Quantum Error Correction.

References

[1] Beatrice Branchini et al. "The Hitchhiker's Guide to FPGA-Accelerated Quantum Error Correction". In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 02. 2023, pp. 338–339. DOI: 10.1109/QCE57702.2023.10271.

[2] Balázs Dezső, Alpár Jüttner, and Péter Kovács. "LEMON–an open source C++ graph template library". In: *Electronic Notes in Theoretical Computer Science*. Vol. 264. Elsevier, 2011, pp. 23–45.

[3] Oscar Higgott and Craig Gidney. "Sparse Blossom: correcting a million errors per core second with minimum-weight matching". In: *arXiv preprint arXiv:2303.15933* (2023).

[4] Vladimir Kolmogorov. "Blossom V: a new implementation of a minimum cost perfect matching algorithm". In: *Mathematical Programming Computation* 1.1 (2009), pp. 43–67.

[5] Namitha Liyanage et al. "Scalable Quantum Error Correction for Surface Codes using FPGA". In: *arXiv preprint arXiv:2301.08419* (2023). URL: https://arxiv.org/pdf/2301.08419.

[6] S. Vittal, P. Das, and M. Qureshi. "Astrea: Accurate quantum error decoding via practical minimum-weight perfect-matching". In: (2023).

[7] Yue Wu and Lin Zhong. "Fusion Blossom: Fast MWPM Decoders for QEC". In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 01. 2023, pp. 928–938. DOI: 10.1109/QCE57702.2023.00107.