POLITECNICO DI TORINO
Repository ISTITUZIONALE

AMARETTO: Enabling Efficient Quantum Algorithm Emulation on Low-Tier FPGAs

Original
AMARETTO: Enabling Efficient Quantum Algorithm Emulation on Low-Tier FPGAs / Conti, C., Volpe, D., Graziano, M., Zamboni, M., Turvani, G.. - (2024), pp. 1-4. (31st IEEE International Conference on Electronics, Circuits and Systems (ICECS) Nancy (FRA) 18-20 November 2024) [10.1109/icecs61496.2024.10848965].

Availability:
This version is available at: 11583/2997142 since: 2025-02-01T11:40:59Z

Publisher:
IEEE

Published
DOI:10.1109/icecs61496.2024.10848965

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

25 November 2026

AMARETTO: Enabling Efficient Quantum Algorithm Emulation on Low-Tier FPGAs

Christian Conti*, Deborah Volpe*, Mariagrazia Graziano*, Maurizio Zamboni*, and Giovanna Turvani*
*Politecnico di Torino Italy
{christian.conti, deborah.volpe, mariagrazia.graziano, maurizio.zamboni, giovanna.turvani}@polito.it

***Abstract*—Researchers and industries are increasingly drawn to quantum computing for its computational potential. However, validating new quantum algorithms is challenging due to the limitations of current quantum devices. Software simulators are time and memory-consuming, making hardware emulators an attractive alternative.**
This article introduces AMARETTO (quAntuM ARchitecture EmulaTion TechnOlogy), designed for quantum computing emulation on low-tier Field-Programmable gate arrays (FPGAs), supporting Clifford+T and rotational gate sets. It simplifies and accelerates the verification of quantum algorithms using a Reduced-Instruction-Set-Computer (RISC)-like structure and efficient handling of sparse quantum gates. A dedicated compiler translates OpenQASM 2.0 into RISC-like instructions. AMARETTO is validated against the Qiskit simulators. Our results show successful emulation of sixteen qubits on a AMD Kria KV260 SoM. This approach rivals other works in emulated qubit capacity on a smaller, more affordable FPGA.

***Index Terms*—Quantum Computing Emulation, Field Programmable Gate Array, Quantum Algorithm Verification, Quantum Computing Simulation,**

I. Introduction

In recent years, interest in **quantum computing** has achieved unique acceleration thanks to its potential in data-intensive applications. Nonetheless, the **validation** of new quantum computing algorithms is challenging due to the constraints imposed by current quantum devices. The production, administration, and upkeep of quantum hardware are exclusive domains of major corporations that grant access through cloud-based platforms, although usually with fees. Furthermore, the fidelity of outcomes can be substantially compromised by devices' noise.

Classical simulation remains the most popular solution for debugging, providing insights into the quantum state, hard to retrieve on real quantum hardware, but **software simulation** faces drawbacks, such as long execution times and high memory requirements, limiting scalability. Hence, exploring **classical hardware platforms** such as **Field-Programmable Gate Arrays (FPGAs)** holds significant promise. Indeed, hardware emulators are expected to outperform software-based counterparts in simulating quantum phenomena due to their ability to replicate the parallel nature of quantum computation more accurately.

This work introduces **AMARETTO** (**quAntuM ARchitecture EmulaTion TechnOlogy**), a **Reduced-Instruction-Set-Computer (RISC)**-like architecture for quantum emulation on **low-tier FPGAs**, supporting **Clifford+T** and **rotational gate sets**. Validated using the Qiskit simulators, AMARETTO successfully emulated **sixteen qubits** on the **AMD Kria KV260 SoM** using a **twenty-bit fixed-point** numeric representation. This approach matches other works' qubit capacity but with a smaller, more accessible FPGA.

The article's organization includes a review of quantum simulation on classical platforms and related work (**Section II**), details of the proposed architecture (**Section III**), results and validation methodology (**Section IV**), and conclusions with future perspectives (**Section V**).

This work was supported in part by AMD under the AMD University program

II. Background and related works

A. Quantum computing emulation

Quantum computing is a new computational paradigm, leveraging quantum mechanic principles like **superposition** and **entanglement**. Its fundamental unit, the **qubit**, can exist in **infinite possible states**, unlike a classical bit which is either 0 or 1. Using Dirac notation, a qubit's state is expressed as the **state vector**:

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle = a\begin{pmatrix}1\\0\end{pmatrix} + b\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}a\\b\end{pmatrix}, \qquad (1)$$

where $|0\rangle$ and $|1\rangle$ are the basis states and a and b are complex **probability amplitudes**. When measured, the qubit collapses to $|0\rangle$ or $|1\rangle$ with probabilities $|a|^2$ and $|b|^2$, respectively.

An n-qubit system's state is represented by the tensor product of individual qubit states:

$$\begin{aligned}|\psi\rangle &= |\psi_{n-1}\rangle \otimes |\psi_{n-2}\rangle \otimes \cdots \otimes |\psi_1\rangle \otimes |\psi_0\rangle = \\ &= c_{0\cdots0}\,|0\cdots0\rangle + \cdots + c_{1\cdots1}\,|1\cdots1\rangle\,,\end{aligned} \qquad (2)$$

Quantum gates, described by **unitary matrices** of dimension $2^m \times 2^m$, where m is the number of involved qubits, modify the system state. Gates involving multiple qubits can create **entanglement**, leading to strong correlations between qubits. To classically simulate a **quantum circuit**, which entails a series of transformations, it is necessary to compute the product of a $2^n \times 2^n$ matrix and the 2^n state vector for each gate layer. These matrices arise from the tensor product of gate matrices for each qubit, assuming the identity matrix when no gate targets a specific qubit, as illustrated in Figure 1. Therefore, the **scalability** challenges due to the **exponential increase in complexity** with the qubits count affecting **operations** and **memory** become evident.

For more details about quantum computing, refer to [1].

B. Previous work

In recent years, various FPGA architectures have emerged to tackle the limitations of software emulation in quantum computing. For instance, [2] introduced an emulator that loads

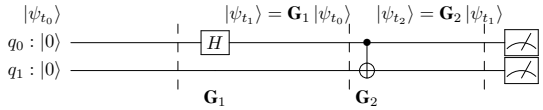


Fig. 1: Example of a two-qubit quantum circuit, highlighting with dotted vertical lines the different layers and showing on the top the state vector evolution layer by layer.

layer matrices from a processor and computes a parallel product with the state vector to determine the new state. However, scalability becomes challenging due to the exponential increase in computation and memory demands with qubit count growth. [3] also employed parallel matrix-vector product technique, enhancing precision by using floating-point number representation but with constraints on the number of emulated qubits. In another work [4], scalability was enhanced by moving the storage of the state vector from the FPGA to external memory.

The approach described in [5] emulates quantum circuits by computing interactions among basis states on an N-dimensional hypercube, reaching the emulation of sixteen qubits.

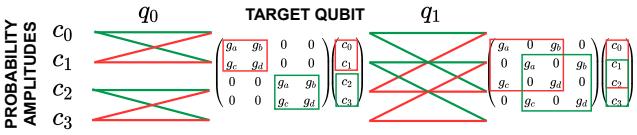


Fig. 2: The butterfly-like mechanism for selecting interacting couples of probability amplitudes in a two-qubit system.

In [6], as in AMARETTO, a **butterfly-like** selection mechanism for interacting couples in the state vector was utilized to reduce unnecessary operations. However, this architecture's area requirement significantly increases with the qubit count and supports only a limited set of gates (Pauli X, CNOT, Toffoli, and Hadamard), restricting its applications. In contrast, AMARETTO supports a universal quantum gate set, enabling the execution of any circuit type.

III. AMARETTO: AN EFFICIENT QUANTUM EMULATOR

AMARETTO (**quAntuM ARchitecture EmulaTion TechnOlogy**) is an efficient architecture for **quantum computing emulation on FPGA** platforms, supporting **Clifford+T and rotational gate sets**, and designed in VHDL for implementation within any modern FPGA by leveraging embedded blocks like Random Access Memories (RAMs) and Digital Signal Processing (DSP) blocks. This **portability** across devices is achieved by modifying the **communication interface** (Figure 4a).

The architecture strategically reduces computational complexity by employing a **butterfly-like** mechanism, as detailed in Section II and shown in Figure 2. This approach isolates interacting probability amplitudes essential for obtaining the output state vector, capitalizing on the sparse nature of equivalent gate matrices. In this way, it is possible to avoid non-operations and the computation of the equivalent layer gate matrix. Two-qubit controlled gates can be implemented by filtering interacting couples associated with the basis state where the control qubit is equal to one. Moreover, a **20-bit fixed-point** number representation (2 bits for decimal and 18 for fractional parts) with a **nearest-even** approximation mechanism is chosen, allowing a reduction of both the area and complexity of arithmetic operators with respect to the floating point one. This also reduces the memory requirements for saving a probability amplitude, leading to more emulable qubits on the same platform. The precision of number representation was chosen based on analysis conducted for the butterfly-based mechanism in [7], which proves that the accuracy of the simulation is not significantly affected by the approximation.

The AMARETTO environment prioritizes user-friendliness (Figure 4b), allowing potential users to describe the quantum circuit using leading quantum frameworks. These generate **OpenQASM 2.0** [8], which is then processed by the **compiler** to translate the gates into a set of supported instructions transmitted to the emulator. Upon completing the simulation process, the user receives the probability amplitudes of the final state vector, providing data ready for user analysis.

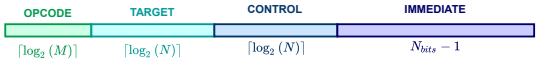
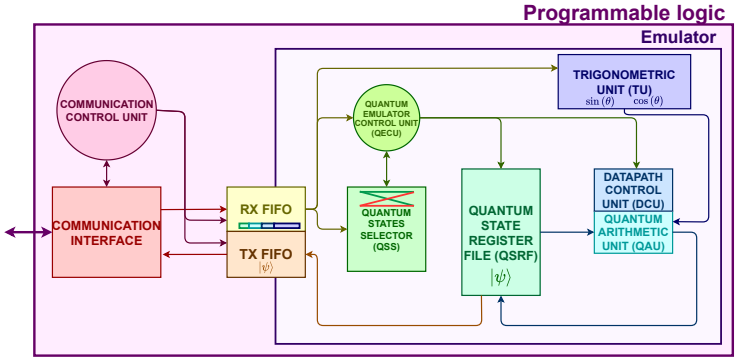


Fig. 3: AMARETTO g-type instruction, separating the fields.

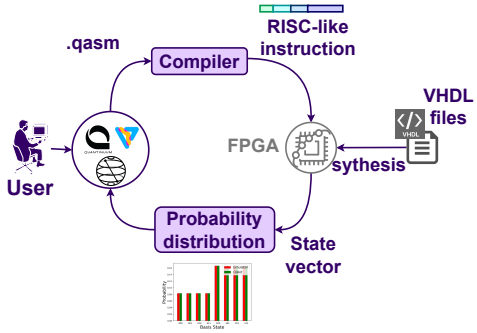
The instructions can be classified into three types: the **s-type** for setting the number of qubits in the circuit, the **g-type** for executing gates, and the **r-type** for reading the state vector. As shown in Figure 3, the g-type instructions include an opcode identifying the gate, bits defining the target and control qubits — in case a single-qubit gate, target and control field coincide —, and an immediate field containing normalized angles. For s-type, the immediate is equal to the number of qubits, while for r-type, only the opcode field is relevant. For the target FPGA, an instruction is 32-bit long since five bits are considered for the opcode, eight bits are for target and control qubits identification (sixteen emulable qubits), and nineteen bits are associated with immediate (an angle represented in the range $[-1, 1)$, with eighteen bits for the fractional part). However, the length of the instruction varies based on both the number of emulable qubits on the target FPGA and the number of fractional bits considered for precision. Furthermore, the compiler indirectly supports additional gates — in particular, all the gates supported by the OpenQASM 2.0 — by leveraging known equivalences in the literature.

AMARETTO follows a **RISC-like** structure (Figure 4a), including a **register file**, which stores the real and imaginary parts of the state vector (**Quantum State Register File**, **QSRF**), the data path for evaluating gate effects on probability amplitude interacting couples (**Quantum Arithmetic Unit**, **QAU**), a **Quantum State Selector** (**QSS**), implementing the butterfly selection mechanism, a **Trigonometric Unit** (**TU**), which computes sine and cosine, a **control unit** (Quantum Emulator Control Unit, **QECU**) and the **communication interface** responsible for receiving architecture instructions and managing the state vector.

The **QSRF**, sized at 2^N elements (where N is the number of



(a) AMARETTO architecture: comprising a register for state vector elements, a state selector executing the butterfly algorithm, a computing unit, a Trigonometric Unit (TU), and a central control unit (QECU).



(b) High-level description of the AMARETTO emulation environment.

Fig. 4: AMARETTO architecture and high-level scheme of its emulation environment.

qubits), optimizes space utilization by taking full advantage of the BRAM blocks, operating with two output and one input ports clocked at double the nominal frequency. This configuration, called **pumping** [9], enables the reading and writing of two probability amplitudes in each clock cycle.

Differently from the previous works, AMARETTO computes **couple by couple the probability amplitudes** to minimize area requirements and increase the number of simulable qubits. The **instruction level parallelism** can be exploited by introducing five pipeline levels. This strategic implementation significantly reduces time penalties, taking advantage of the absence of data dependencies in the execution of a single gate since the interacting couples are independent of each other. The pipeline reaches its maximum effectiveness when the number of couples to update is equal to or higher than the number of pipeline stages, i.e., when $2^{N_q-2} \geq N_{\text{pipe}} \rightarrow N_q \geq \lceil \log_2(N_{\text{pipe}}) + 2 \rceil = N_{q_{\min}}$, where N_q represents the number of qubits in the circuit and N_{pipe} denotes the number of pipeline stages (five in this context). This is because the execution of two consecutive gates presents data dependencies. Consequently, for circuits with qubits count lower than $\lceil \log_2(N_{\text{pipe}}) + 2 \rceil$, stalls must be inserted to ensure the correctness of the results. For saving area, it was decided to compute the update of at least $2^{N_{q_{\min}}}$ couples also for smaller circuits but not to store the outcomes exceeding 2^{N_q}. This approach eliminates the need to instantiate a dedicated unit to manage stalls while maintaining the same time penalty.

The pipeline can be introduced by standardizing the execution of the supported gates and recognizing that all can be implemented as:

$$\begin{aligned} c_{i_{\text{out}}} &= \alpha \sin(\theta) + \beta \cos(\theta) + i(\gamma \sin(\theta) + \delta \cos(\theta)) \\ c_{j_{\text{out}}} &= \epsilon \sin(\theta) + \zeta \cos(\theta) + i(\eta \sin(\theta) + \iota \cos(\theta)), \end{aligned} \tag{3}$$

where α, β, γ, δ, ϵ, ζ, η and ι are properly chosen depending on the gate for selecting real or imaginary parts of the probability amplitudes in input, $c_{i_{\text{out}}}$ and $c_{j_{\text{out}}}$ are the couple of probability amplitudes associated with i^{th} and j^{th} basis states in the output state vector and θ — immediate field of the instruction — is the parametric angle in the rotational gate and a gate-dependent fixed angle in the others. The sine and cosine values are exploited to change the sign or delete a factor, thanks to trigonometric properties. Therefore, the datapath comprises four computing units — one for the real, one for the imaginary part of each probability amplitude —, containing two multipliers and an adder. Sine and cosine are computed by the **TU**.

TABLE I: Comparison between AMARETTO synthesis results and the current literature.

Emulator	**AMARETTO**	[2]	[3]	[4]	[5]	[6]
N_{qubit}	16	2	4	32	16	9
Device	AMD Kria KV260	Intel Cyclone V	Intel Arria 10	Intel Arria 10	Intel(APEX) 20KE1500	Intel Stratix
BRAM	2.62 MB	-	32.08 MB	32 GB (ext.)	-	-
Logic Utilization	7751/117120 CLB	8000 ALMs	374021 ALMs	56219 ALMs	1500000* Gates	4019 LC
DSP	11/1248	-	1364	49	-	-
Precision	20-bit fixed	10-bit fixed	32-bit float	64-bit float	-	18-bit fixed
f_{clk} **[MHz]**	100 MHz	-	233 MHz	233 MHz	60 MHz	-

The **TU** implements the architecture presented in [10], which proposed an efficient approach based on the exploitation of look-up tables (LUTs) and the Taylor series. This solution should help the target application achieve a **better balance between area and accuracy**, compared to a solution based on the COordinate Rotation DIgital Computer (CORDIC) algorithm. Although CORDIC can be fully unrolled, the considered approach involves fewer processing elements. All the emulator blocks are synchronized and managed by the **QECU**.

The interaction with the external is implemented through **asynchronous First-In-First-Out** (**FIFO**) buffers (platform independent) and a **communication interface unit** (platform dependent), both coordinated by a communication control unit. The two buffers, one for transmission of the state vector and the other for the reception of the instruction to execute, handle the **clock domain crossing** to avoid metastability issues. The communication interface unit is the only thing that should be modified varying the platform. In this context, it implements the AMBA 4 AXI4-Stream communication ARM protocol, permitting the exploitation of the Direct Memory Access (DMA) mechanism.

IV. Results

The architecture synthesis on the **AMD Kria KV260 SoM** using **Vivado 2023.1** achieved a maximum of **sixteen qubits**. **RAM availability** emerged as the **bottleneck**, reaching 100% utilization. This aligns with expectations as memory dominates due to its $\mathcal{O}(N_{\text{bit}}2^{N_q})$ scaling, where N_{bit} is the number of bits exploited for numerical representation.

The obtained synthesis results are compared with the current literature in Table I. Making direct comparisons poses challenges due to differences among various architectures in the different target FPGAs, produced by different companies. Furthermore, the reported information in these articles is often incomplete, and the synthesis of many of these architectures depends on the quantum circuit, differently from AMARETTO. Indeed, its advantage lies in **not requiring re-synthesis** for executing new circuits, unlike other architectures. Additionally, its **extensive gate support** allows it to handle all applications below the platform's maximum capacity efficiently.

However, some observations can still be made. Although the board considered in this work is relatively small, [4] is the only architecture achieving a higher number of emulable qubits leveraging external memory. Nevertheless, [4] architecture demands relatively more memory in proportion to the number of qubits with respect to AMARETTO, as it employs more bits for number representation, storing both the state vector and the gate matrices.

Additionally, the relative logic occupation of our architecture is the lowest among the compared designs. Furthermore, despite operating at a lower frequency than [3], [4], AMARETTO is expected to be faster since a single gate execution requires $\mathcal{O}(N)$ clock periods instead of $\mathcal{O}(N^2)$, where N is the length of the state vector, i.e. 2^{N_q}.

Functional verification involved about **fifty quantum circuits** in OpenQASM 2.0, compared with Qiskit's state vector simulator using **Great-circle distance (GCD)**, thus considering state vector elements in polar coordinate, i.e. as points on a sphere, and their spheric distance estimates the divergence between the two results. The GCD accentuates the differences between complex numbers, thus guaranteeing a more reliable functional validation of the architecture. GCD consistently remained below 0.05, meeting the study's acceptability threshold.

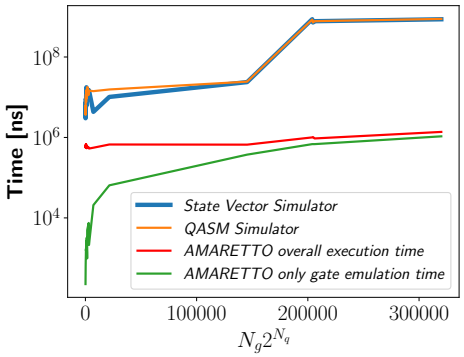


Fig. 5: Comparison of execution time between Qiskit simulators and AMARETTO, showing hardware emulation's significant advantage over software.

Figure 5 compares execution times of Qiskit simulators (Qasm and State Vector) — on a single-process Intel(R) Xeon(R) Gold 6134 CPU @ 3.20 GHz opta-core, Model 85, with a memory of about 103 GB [11] — and AMARETTO, demonstrating hardware emulation's **orders of magnitude lower** time requirements, particularly evident with larger quantum circuits. The execution time scales as $\left(2^{\max(N_q,N_{q_{\min}})-1}\frac{N_g(2-\alpha)}{2}+(N_{\text{pipe}}-1)\right)T_{\text{clock}}$, where α is the percentage of controlled gates and T_{clock} the clock period. Therefore, it scales linearly with $2^{N_q}N_g$.

V. Conclusions

This article introduces AMARETTO, a specialized architecture for quantum computing emulation on low-tier FPGAs, supporting Clifford+T and rotational gate sets. A comparative analysis shows AMARETTO's execution time is about two orders of magnitude faster than the Qiskit state vector simulator. Emulating sixteen qubits on a AMD Kria KV260 SoM, AMARETTO matches the qubit capacity of other works using a smaller and cheaper FPGA. This promising solution addresses challenges in validating new quantum algorithms, potentially advancing quantum computing application development.

References

[1] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge ; New York: Cambridge University Press, 10th anniversary ed ed., 2010.

[2] J. Pilch and J. Długopolski, "An fpga-based real quantum computer emulator," *Journal of Computational Electronics*, vol. 18, pp. 329–342, 2019. https://doi.org/10.1007/s10825-018-1287-5.

[3] N. Mahmud and E. El-Araby, "A scalable high-precision and high-throughput architecture for emulation of quantum algorithms," in *2018 31st IEEE International System-on-Chip Conference (SOCC)*, pp. 206–212, IEEE, 2018. https://doi.org/10.1109/SOCC.2018.8618545.

[4] N. Mahmud, B. Haase-Divine, A. Kuhnke, A. Rai, A. MacGillivray, and E. El-Araby, "Efficient computation techniques and hardware architectures for unitary transformations in support of quantum algorithm emulation," *Journal of Signal Processing Systems*, vol. 92, pp. 1017–1037, 2020. https://doi.org/10.1007/s11265-020-01569-4.

[5] M. Fujishima, K. Saito, and K. Hoh, "16-qubit quantum-computing emulation based on high-speed hardware architecture," *Japanese Journal of Applied Physics*, vol. 42, no. 4S, p. 2182, 2003. https://doi.org/10.1143/JJAP.42.2182.

[6] C. Conceição and R. Reis, "Efficient emulation of quantum circuits on classical hardware," in *2015 IEEE 6th Latin American Symposium on Circuits & Systems (LASCAS)*, pp. 1–4, IEEE, 2015. https://doi.org/10.1109/ICCD.2004.1347938.

[7] M. L.Lagostina, M.Zamboni and G.Turvani, "Aequam, a fast and efficient quantum emulation toolchain," 2022. https://webthesis.biblio.polito.it/25427/.

[8] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," *arXiv preprint arXiv:1707.03429*, 2017. https://doi.org/10.48550/arXiv.1707.03429.

[9] A. M. Abdelhadi and G. G. Lemieux, "Modular multi-ported sram-based memories," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '14, (New York, NY, USA), p. 35–44, Association for Computing Machinery, 2014. https://doi.org/10.1145/2554688.2554773.

[10] F. De Dinechin, M. Istoan, and G. Sergent, "Fixed-point trigonometric functions on fpgas," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 5, pp. 83–88, 2014. https://doi.org/10.1145/2641361.2641375.

[11] "Intel Xeon Gold 6134 processor - product specification." [Online] https://ark.intel.com/content/www/us/en/ark/products/120493/intel-xeon-gold-6134-processor-24-75m-cache-3-20-ghz.html, accessed 17-November-2022.