Towards Quantum Circuit Emulation on Low-Tier FPGAs

Towards Quantum Circuit Emulation on Low-Tier FPGAs

Christian Conti*, Deborah Volpe *, Giovanni Amedeo Cirillo*
, Mariagrazia Graziano†, Maurizio Zamboni*, and Giovanna Turvani*
*Department of Electronics and Telecommunications, Politecnico di Torino Italy
†Department of Applied Science and Technology, Politecnico di Torino Italy
{christian.conti, deborah.volpe, giovanni_cirillo, 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 current quantum device limitations. Software simulators are time and memory-intensive, making hardware emulators an attractive alternative.**

This article introduces a digital architecture, designed to emulate quantum computing on low-tier Field-Programmable Gate Arrays (FPGAs), supporting Clifford+T and rotational gate sets. It simplifies and accelerates quantum algorithm verification using a RISC-like structure and efficiently handling sparse quantum gates. A dedicated compiler translates OpenQASM 2.0 into RISC-like instructions. The architecture is validated against the Qiskit state vector simulator, successfully emulating sixteen qubits on a Xilinx Kria KV260 SoM.

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

I. Introduction

Interest in **quantum computing** (**QC**) has grown in recent years. However, current quantum devices face limitations due to ideal phenomena as decoherence and relaxation, that affect the execution of a quantum application. This implies that it would be better to perform a preliminary less-scaled validation of a quantum algorithm on noiseless classical **simulators**, capable of providing insights into the inner qubits states. This **classical** validation can be done either through software simulation or hardware emulation, with hardware emulation offering significant advantages in speed and resource efficiency.

This work describes a quantum circuits emulator for low-tier Field Programmable Gate Arrays (FPGAs), interfaceable with existing quantum computing frameworks through an Open QASM 2.0 parser.

II. Methodology and Implementation

The proposed architecture for **quantum computing emulation on FPGA** platforms supports **Clifford+T and rotational gate sets**. It is portable across any modern Field Programmable Gate Arrays (FPGAs) by modifying the communication interface, shown in Figure 2, since the design uses only commonly-available embedded blocks like Random Access Memories (RAMs) and Digital Signal Processing (DSP) block. The architecture strategically reduces computational complexity by employing a **butterfly-like** mechanism [1], which reduces unnecessary operations by exploiting the sparse nature of gate matrices. In particular, it isolates interacting probability amplitudes essential for obtaining the output state vector, as shown in Figure 1. The two-qubit control gates are implemented by selectively filtering interacting couples associated with the basis state where the control qubit is equal to one.

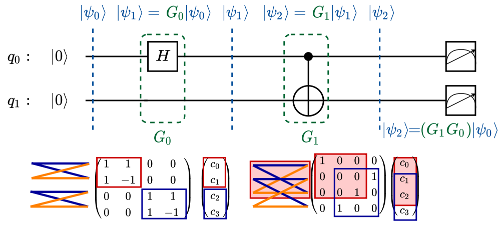
$|\psi_0\rangle$
$|\psi_1\rangle = G_0|\psi_0\rangle$
$|\psi_1\rangle$
$|\psi_2\rangle = G_1|\psi_1\rangle$
$|\psi_2\rangle$
$q_0 :$
$|0\rangle$
$q_1 :$
$|0\rangle$
H
G_0
G_1
$|\psi_2\rangle=(G_1G_0)|\psi_0\rangle$
$\begin{pmatrix}1 & 1 & 0 & 0\\ 1 & -1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 0 & 0 & 1 & -1\end{pmatrix}$
$\begin{pmatrix}c_0\\ c_1\\ c_2\\ c_3\end{pmatrix}$
$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\end{pmatrix}$
$\begin{pmatrix}c_0\\ c_1\\ c_2\\ c_3\end{pmatrix}$

Fig. 1: The butterfly-like mechanism for selecting interacting couples of probability amplitudes in a two-qubit system, considering the Bell state generator circuit.

To reduce the area and complexity of arithmetic operators, a **20-bit fixed-point** number representation (2 bits for the integer part and 18 bits for the fractional part) with a nearest-even rounding mechanism is chosen instead of **floating-point** representation. This approach also offers significant benefits in terms of memory requirements. The impact of fixed-point representation on result accuracy has been evaluated in [2], demonstrating that the approximation does not significantly affect the accuracy.

The architecture is integrated into an environment that prioritizes user-friendliness, enabling users to describe quantum circuits using major quantum frameworks such as **Qiskit** (Figure 2). These frameworks generate **OpenQASM 2.0**, which is then processed by the compiler to translate the gates into a set of supported instructions transmitted to the emulator. Users receive the probability amplitudes of the final state vector.

The architecture follows a RISC-like structure (Figure 2) and includes several key components:

- a register file (**Quantum State Register File**) for storing the real and imaginary parts of the state vector operating with two-output and one-input ports clocked at double the architecture's operating frequency to use a single input port as two write port;
- a **Quantum Arithmetic Unit** (**QAU**) for computing the impact of gates on probability amplitude interacting pairs;
- a **Quantum State Selector** (**QSS**), which implements the butterfly selection mechanism;
- a **Trigonometric Unit** (**TU**) for computing sine and cosine values exploiting the architecture presented in [3];
- a control unit (**Quantum Emulator Control Unit**);

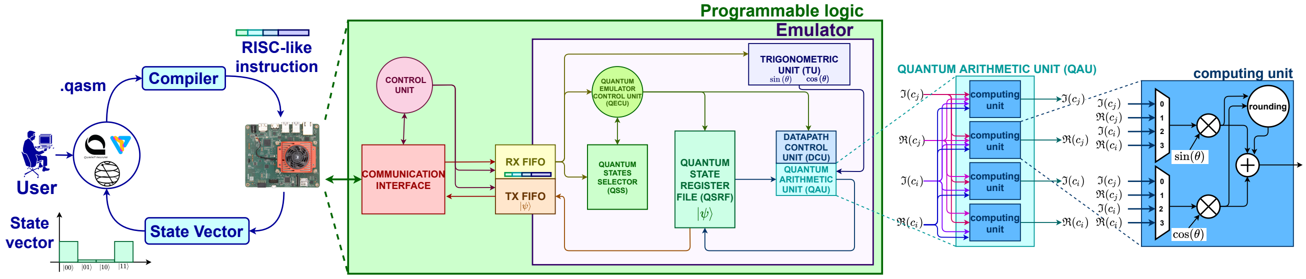

Fig. 2: Architecture: comprising a register for state vector elements, a state selector executing the butterfly algorithm, a computing unit (QAU), a Trigonometric Unit (TU), and a central control unit (QECU).

- a **communication interface** responsible for receiving instructions and managing the state vector.

Unlike other state-of-the-art approaches, our architecture calculates probability amplitudes for each interacting pair of qubits, minimizing area requirements and increasing the number of simulable qubits. By introducing five pipeline stages, we exploit instruction-level parallelism (Figure 3), significantly reducing time penalties due to the absence of data dependencies between independent interacting pairs. The pipeline reaches maximum efficiency when the number of interacting pairs is equal to or greater 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 (which is 5 in this context). For circuits with fewer qubits, stalls must be inserted to ensure correct results.

To save area, we compute updates for N_{pipe} pairs in smaller circuits but do not store outcomes exceeding 2^{N_q}. This approach avoids the need for a dedicated stall management unit while maintaining the same time penalty.

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

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

where α, β, γ, δ, ϵ, ζ, η and ι are coefficients determined by the gate, selecting real or imaginary parts of the input probability amplitudes. $c_{i_{\text{out}}}$ and $c_{j_{\text{out}}}$ are the probability amplitudes associated with the i^{th} and j^{th} basis states in the output state vector, and θ is the parametric angle for rotational gates or a fixed angle for other gates. The sine and cosine values change signs or eliminate factors using trigonometric properties.

Thus, the datapath includes four computing units—one for each real and imaginary part of the probability amplitudes—each containing two multipliers and an adder (Figure 2). The trigonometric unit computes sine and cosine values.

III. Results and Conclusions

Synthesizing on the **Xilinx Kria KV260 SoM** using Vivado **2023.1** with default directives achieved a qubit count of **sixteen**. The architecture's bottleneck is RAM availability, reaching 100% utilization. In contrast, the configurable logic block (CLB) occupancy was only 6.62%, and the Digital Signal Processing (DSP) resources usage was 0.88%, excluding the communication block. This aligns with expectations, as memory utilization scales as $\mathcal{O}(N_{\text{bit}} 2^{N_q})$, where N_{bit} is the number of bits used for number representation. Meanwhile, the TU and QAU scale as $\mathcal{O}(N_{\text{bit}})$, and the QSS scales as $\mathcal{O}(N_q)$. Thus, memory occupation is the dominant factor.

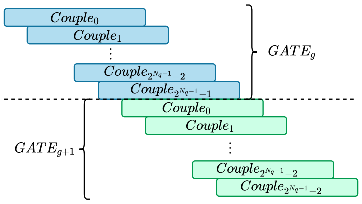

Fig. 3: Pipelined gate execution.

The architecture was verified using fifty OpenQASM 2.0 quantum circuits from public repositories involving up to sixteen qubits, comparing the resulting state vector with the Qiskit state vector simulator output. The comparison used the Great-circle distance (GCD) in polar coordinates to measure divergence. The GCD remained consistently below 0.05 in all tests, meeting the study's acceptability threshold.

The execution time of the architecture 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 N_{pipe} is the number of pipeline stages (five in this case), α is the percentage of controlled gates and T_{clock} is the clock period. This scales linearly with $2^{N_q} N_g$. The execution time is **orders of magnitude lower** than Qiskit software emulators executed 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, proving the advantage of hardware emulation, with benefits increasing with circuit size.

Acknowledgements

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

References

[1] G. Negovetic, M. Perkowski, M. Lukac, and A. Buller, "Evolving quantum circuits and an fpga-based quantum computing emulator," 2002. https://pdxscholar.library.pdx.edu/ece_fac/191/.

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

[3] 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.