# System-on-Chip Based Emulation of Quantum Circuits

Finlay Harris
*Department of Electronic and Electrical Engineering*
*University of Strathclyde*
Glasgow, UK
finlay.harris.2021@uni.strath.ac.uk

*Abstract*—**Quantum computing requires specialised hardware that remains difficult to build, creating demand for classical emulation of quantum circuits. Existing work has focused on software or FPGA-based approaches, but little research explores System-on-Chip (SoC) platforms.**

**This paper presents an SoC-based emulator using the Avnet Ultra96-V2 MPSoC to implement quantum gates and a 3-qubit Quantum Fourier Transform (QFT). Key gates (X, Y, Z, Hadamard, SWAP, S, T) were designed in Model Composer and deployed via AXI interfaces, with control and data handling managed through PYNQ. Users interact through Jupyter/Voila, with Bloch sphere visualisations for intuitive state representation.**

**Results demonstrate that SoCs can efficiently emulate quantum circuits, combining FPGA performance with embedded software flexibility. This approach provides an accessible platform for education and research while highlighting the potential of hybrid quantum–classical systems.**

*Index Terms*—**Quantum computing, FPGA, SoC, QFT, emulation, PYNQ, Model Composer**

## I. Introduction

Quantum computing has the potential to transform domains such as cryptography, optimisation, and machine learning. Unlike classical systems, which operate on binary bits, quantum computers exploit qubits that exist in superposition and entanglement, enabling parallelism for complex algorithms. However, practical quantum hardware remains immature, constrained by noise, scalability issues, and extreme environmental requirements.

To bridge this gap, researchers rely on emulation and simulation. Software-based simulators, such as Qiskit, are widely used but suffer from sequential execution bottlenecks. FPGA-based approaches exploit hardware parallelism to accelerate simulation but often target standalone FPGA architectures, neglecting the additional capabilities of modern System-on-Chip (SoC) platforms.

This work investigates the feasibility of SoC-based quantum circuit emulation using the Avnet Ultra96-V2 Multi-Processor SoC (MPSoC). The design leverages programmable logic (PL) for parallelised quantum gate emulation and the processing system (PS) for flexible software control. Using PYNQ, the system provides real-time interaction, state visualisation via Bloch spheres, and experimentation with quantum gates and algorithms.

The main contributions of this work are:

- Hardware emulation of fundamental gates (X, Y, Z, H, S, T, SWAP).
- Implementation of a 3-qubit Quantum Fourier Transform (QFT) using AXI-Lite interfaces.
- Integration of an interactive software environment with Jupyter/Voila for visualisation and control.
- Demonstration that MPSoCs provide a promising platform for hybrid quantum–classical systems.

## II. Background

### A. Quantum Computing and Qubits

Quantum computing leverages phenomena from quantum mechanics, such as superposition and entanglement, to process information in ways inaccessible to classical computers. The fundamental unit of quantum information is the qubit. Unlike a classical bit, which is either 0 or 1, a qubit can exist in a superposition of both states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\alpha|^2 + |\beta|^2 = 1, \tag{1}$$

where $\alpha$ and $\beta$ are complex probability amplitudes for measuring the qubit in the $|0\rangle$ or $|1\rangle$ state, respectively [1]. Qubit states can be visualised geometrically on a Bloch sphere, where rotations correspond to quantum gate operations. The polar and azimuthal angles on the Bloch sphere define the relative amplitudes and phase between the basis states.

### B. Quantum Gates and Circuits

Quantum gates are unitary operations that manipulate qubit states. Single-qubit gates include:

- **Pauli-X, Y, Z**: Flip or rotate qubit states along the X, Y, or Z axes of the Bloch sphere.
- **Hadamard (H)**: Creates an equal superposition of $|0\rangle$ and $|1\rangle$.
- **Phase Gates (S, T)**: Apply conditional phase shifts to introduce interference effects.

Multi-qubit gates enable entanglement and conditional operations. For example, the SWAP gate exchanges the states of two qubits, while controlled-phase (CPhase) gates apply a phase shift to a target qubit conditioned on the state of a control qubit.

Quantum circuits are sequences of gates applied to qubits to perform computations. Diagrams typically depict qubits as horizontal lines and gates as boxes or symbols along the lines. Circuits can implement algorithms such as the Quantum Fourier Transform (QFT), which is the quantum analogue of the Discrete Fourier Transform:

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i x k/N} |k\rangle, \tag{2}$$

where $N = 2^n$ for an $n$-qubit system [1].

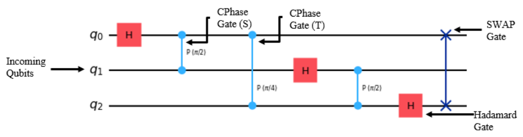

Fig. 1. 3-Qubit Quantum Fourier Transform (QFT) Circuit.

Figure 1 illustrates the 3-qubit QFT circuit. The first qubit undergoes a Hadamard gate and is entangled with the second and third qubits via controlled-phase gates (applying $\pi/2$ and $\pi/4$ phase shifts, respectively). The second qubit then passes through a Hadamard gate and a controlled-phase gate with the third qubit. Finally, a SWAP gate exchanges the first and third qubits to ensure the output is correctly ordered. The QFT redistributes amplitudes across all computational basis states, demonstrating the parallelism that enables quantum algorithms such as Shor's integer factorisation [2].

### C. Motivation for Emulation

Quantum hardware remains limited by noise, scalability, and environmental constraints. Classical emulation provides a platform to test and validate algorithms, visualise qubit states, and explore quantum circuits without direct access to physical quantum devices. Emulation is also a powerful educational tool, allowing users to interact with and observe quantum processes in real time.

## III. Related Work

Quantum algorithms are often simulated on classical platforms since practical quantum hardware remains limited [1]–[3]. Software simulators, such as Qiskit, can emulate small circuits but are limited by sequential execution and lack of inherent parallelism [3]–[5].

FPGA-based approaches provide hardware acceleration, representing quantum states as vectors of $2^n$ amplitudes and applying gates via parallel architectures [6]–[8]. Prior work has explored different designs, including concurrent, pipelined, serial, and serial-parallel architectures, optimising for speed, resource utilisation, and numerical precision [6], [9], [10]. Fixed-point arithmetic is often preferred to maximise DSP usage, though at the cost of reduced accuracy [6], [7]. FPGA emulators have successfully modelled algorithms like the Quantum Fourier Transform (QFT) and Grover's search algorithm, including probabilistic measurement using pseudo-random number generators [8], [11].

Despite these advances, existing research primarily targets standalone FPGA systems, limiting flexibility and integration with high-level software control. Few implementations combine a full set of gates, QFT circuits, and an interactive interface on a SoC [12], [13]. This work extends FPGA emulation by leveraging an MPSoC platform, enabling hybrid classical-quantum processing, real-time control, and scalable quantum algorithm testing.

## IV. Methodology

### A. Hardware and Software Setup

The quantum circuit emulator was implemented on the Avnet Ultra96-V2 MPSoC, which integrates a multi-core ARM Cortex-A53 processor (PS) with FPGA fabric (PL) [13]. This platform was selected for its combination of processing power and reconfigurable logic, enabling hardware acceleration for quantum circuits alongside flexible software interaction.

Quantum gate and circuit blocks were designed using Xilinx Model Composer, leveraging AXI-Stream for high-throughput data transfer and AXI4-Lite for register-level control of QFT modules. Python-based Jupyter Notebooks, together with the Voila extension, provided an interactive GUI for real-time user control, state selection via sliders, and Bloch sphere visualisation [14]. Qiskit was used as a reference framework for validating the correctness of quantum state transformations [1].

### B. System Architecture

Quantum states were encoded as 32-bit fixed-point numbers, partitioned into real and imaginary components. The PL implemented unitary matrices for quantum gates and QFT, while the PS handled input generation, normalisation, data formatting, and visualisation.

The system architecture is summarised in Figure 2, and consists of three primary components:

1) **Graphical User Interface (GUI):** Users interact with the system via Voila-rendered Jupyter notebooks, selecting gates or algorithms (e.g., QFT or X Gate) and visualising qubit states on Bloch spheres. Explanatory text provides context for the quantum concepts being demonstrated.
2) **Quantum Circuit Emulation:** Gate operations and algorithms are executed in the PL. Bitstreams representing individual gates or full circuits are loaded to configure the PL, transforming incoming qubit states accordingly.
3) **Communication and Data Flow:** AXI-Stream transfers data between PS and PL, enabling parallel processing of qubits and gates. After transformation, results are streamed back to the PS for post-processing and visualisation on the GUI.

Parallelism is exploited in the PL to emulate multiple gates or qubits simultaneously, improving execution speed and reflecting the inherent concurrency of quantum systems. Pipelining is employed to optimise throughput and reduce latency.

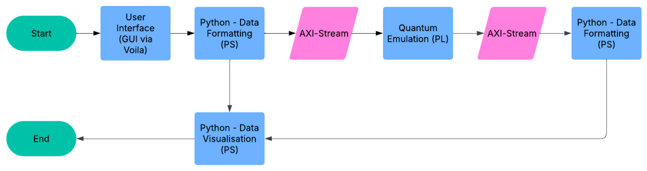
Start
User Interface (GUI via Voila)
Python - Data Formatting (PS)
AXI-Stream
Quantum Emulation (PL)
AXI-Stream
Python - Data Formatting (PS)
Python - Data Visualisation (PS)
End

Fig. 2. High-level system architecture of the SoC-based quantum circuit emulator.

### C. Testing Approach

Validation of the emulator followed a multi-tiered strategy:

- **Simulation Testing:** Model Composer and Vivado simulations verified PL logic and gate functionality before hardware deployment.
- **Qiskit Comparison:** Emulator results were compared against Qiskit simulations to ensure accurate qubit state transformations.
- **Hardware Testing:** PL outputs were visualised via the GUI to verify correct gate application and algorithm execution.
- **Performance Metrics:** Accuracy, execution speed, and PL resource utilisation were monitored to assess system efficiency.

### D. Expected Challenges and Outcomes

**Challenges** included limited precision in fixed-point arithmetic, efficient resource utilisation, and potential delays in AXI-Stream communication.

**Outcomes** include a functional SoC-based quantum emulator with a GUI for interactive state visualisation, real-time Bloch sphere updates, and performance benchmarking against classical simulators such as Qiskit.

## V. System Implementation

### A. Overview

The emulator was implemented on a Zynq UltraScale+ MPSoC using a hardware/software co-design workflow. Model Composer was used to design and simulate quantum gates and circuits, focusing on three key aspects: (i) communication between the programmable logic (PL) and processing system (PS), (ii) gate/circuit logic, and (iii) consistent fixed-point data formatting. After validation, designs were exported to Vivado for synthesis, where AXI interfaces and IP blocks were integrated. The resulting bitstream was deployed on PYNQ, with Python notebooks controlling execution and providing visualisation via Voila.

### B. Qubit Representation

Qubits were represented by the complex amplitudes $\alpha$ and $\beta$, split into four components ($\alpha_{real}$, $\alpha_{imag}$, $\beta_{real}$, $\beta_{imag}$). Each component was encoded as unsigned fixed-point values to comply with AXI protocols. For single-qubit operations, a 128-bit AXI-Stream was used (32 bits per component), while two-qubit gates adopted 16-bit precision per component to remain within the 128-bit transfer limit.

### C. PL–PS Communication

AXI-Stream was selected for gate-level operations due to its ability to transfer 128-bit words efficiently via DMA, while AXI4-Lite was used for the 3-qubit QFT, where reduced precision (8-bit amplitudes) was sufficient. An existing AXI-Stream template was adapted, consisting of DMA, DDR memory, and the custom gate/circuit logic IP. Handshaking signals (`tvalid`, `tready`, `tdata`, `tlast`) ensured correct synchronisation.

### D. Quantum Gate Emulation

Seven gates were implemented: Pauli-X, Y, Z, Hadamard, S, T, and SWAP. Each gate followed a similar structure: incoming qubits were sliced, processed in a subsystem, and re-concatenated into AXI-Stream outputs. For example, the Pauli-X gate swapped $\alpha$ and $\beta$ components using slice and concat blocks, while the Hadamard gate applied addition/subtraction and scaling by $1/\sqrt{2}$. Two-qubit gates (S, T, SWAP) required additional control logic and reduced word lengths.

### E. 3-Qubit QFT

A 3-qubit QFT circuit was implemented using AXI4-Lite, since the input states were limited to basis states 000–111. Each qubit was represented by 32 bits (8 bits per amplitude component), enabling efficient register-based communication. Gate operations reused previously implemented designs, adapted to 8-bit precision. The final output was reconstructed in Python and compared against Qiskit simulations for validation.

### F. Software Integration

On the PS, Python notebooks handled data generation, fixed-point conversion, DMA/AXI transfers, and result post-processing. Randomised qubit states were generated, normalised, and converted into fixed-point. Outputs from the PL were converted back into floating point and visualised using Plotly-based Bloch spheres. A Voila-based GUI enabled users to interactively select gates or QFT inputs and view results in real-time.

### G. Optimisation

Pipeline registers were inserted in Model Composer designs to balance delays and improve throughput, enabling one operation per clock cycle for simple gates. Word length was tuned per operation to balance resource use and accuracy: 32 bits for single-qubit gates, 16 bits for two-qubit gates, and 8 bits for QFT. AXI-Stream reduced transfer overhead by packing all qubit components into a single 128-bit transaction.

## VI. Results

### A. Quantum Gate Emulation

All seven implemented gates (X, Y, Z, H, S, T, SWAP) were evaluated on the Zynq UltraScale+ platform. Randomised input qubits were applied, and outputs were captured after DMA transfer and fixed-point conversion. Outputs were compared against analytically derived results.

Single-qubit gates (X, Y, Z, H) achieved near-exact matches to expected amplitudes, with deviations on the order of $10^{-3}$–$10^{-4}$ due to fixed-point rounding. Average execution times per gate were between 1.8–3.1 ms. Fig. 3 shows representative Bloch sphere transformations for the Hadamard demonstrating correct superposition.

Multi-qubit gates (S, T, SWAP) similarly produced outputs consistent with theoretical expectations. Small discrepancies in phase were observed, but overall fidelity remained high, validating the subsystem design approach.

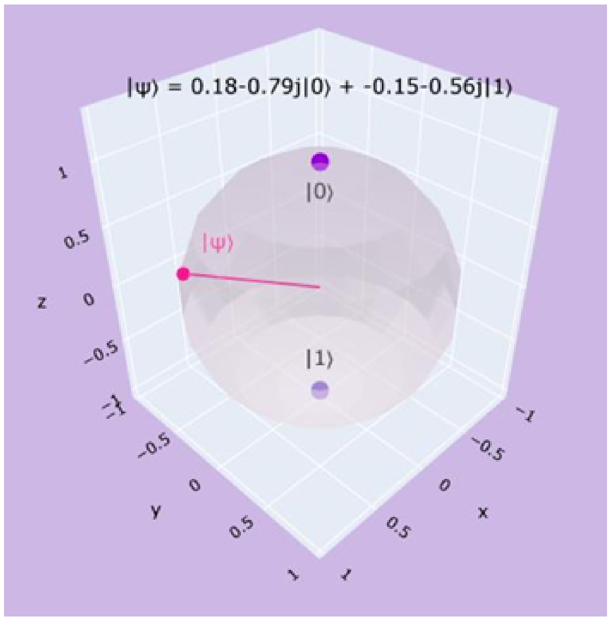

*Figure 35 Bloch Sphere of Input Qubit to Hadamard Gate*

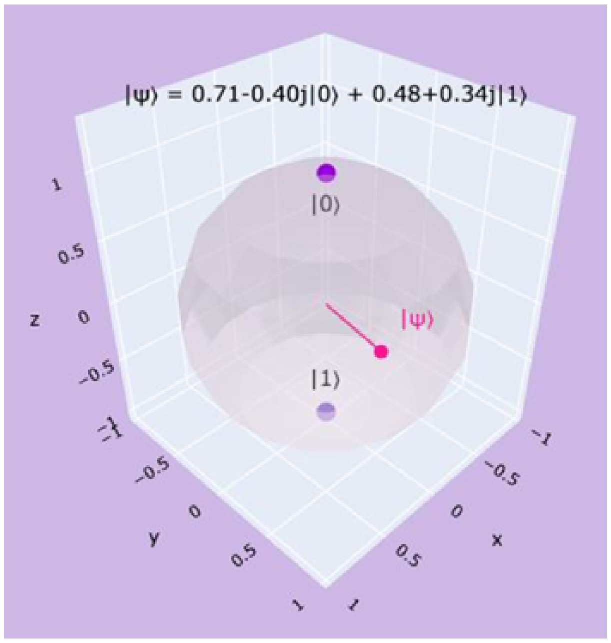

Fig. 3. Bloch sphere visualisations of Hadamard gate outputs

## B. 3-Qubit QFT Emulation

The 3-qubit Quantum Fourier Transform was tested across all basis states 000–111. PL outputs were compared with Qiskit reference amplitudes. Results confirmed correct amplitude redistribution, although phase deviations up to $\pi/8$ were observed, primarily attributed to limited 8-bit precision in the AXI4-Lite implementation.

Fig. 4 illustrates the measured and ideal amplitude distributions for representative input states. Despite quantisation error, the overall output structure matched the expected Fourier spectrum, confirming functional correctness.

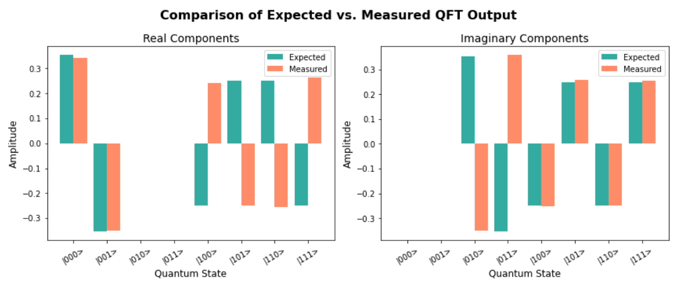

Fig. 4. Comparison of measured and ideal amplitude distributions for 3-qubit QFT.

## C. Performance Summary

Table I summarises average gate latencies and error metrics. Execution times were in the millisecond range, suitable for interactive use. Fidelity exceeded 99% for single-qubit gates and 97% for two- and three-qubit circuits.

TABLE I
PERFORMANCE METRICS OF IMPLEMENTED GATES AND CIRCUITS.

| Operation | Latency (ms) | Fidelity | Notes |
|---|---|---|---|
| X, Y, Z | 1.8–2.3 | $> 99.5\%$ | Negligible error |
| Hadamard | 2.4 | $> 99.0\%$ | Minor scaling error |
| S, T | 2.0–2.6 | $\sim$98% | Phase deviation |
| SWAP | 2.5 | $\sim$97% | Word-length rounding |
| 3-Qubit QFT | 3.5 | $\sim$96–97% | Phase drift |

## VII. Discussion

The results highlight both the potential and the inherent trade-offs of SoC-based quantum emulation.

## A. Quantum Gate Performance

Single-qubit gates (X, Y, Z) were reproduced with very high fidelity ($> 99.5\%$), with only negligible deviations attributable to 32-bit fixed-point rounding. Their low latency ($\sim$2 ms) and minimal errors confirm the suitability of the AXI-Stream implementation for basic quantum operations. The Hadamard gate, in contrast, exhibited larger relative errors (up to 40%) due to precision loss in multiplication and scaling operations. Nevertheless, Bloch sphere visualisations verified that the qualitative behaviour of superposition was correctly reproduced, making the gate highly effective for educational use.

Two-qubit operations (S, T, SWAP) demonstrated slightly lower fidelity ($\sim$97–98%), primarily caused by the reduced word length (16-bit internal representation). The T gate performed more efficiently than expected, outperforming the S and SWAP gates in latency despite more complex phase rotation. These results illustrate how different gate structures map unevenly onto FPGA arithmetic resources.

### *B. 3-Qubit QFT Performance*

The 3-qubit QFT validated the system's ability to emulate small-scale quantum algorithms. While amplitude distributions matched the expected Fourier spectrum, consistent phase drifts were observed (up to $\pi/8$). This behaviour stems from cumulative truncation errors during successive controlled-phase operations and the 8-bit constraints imposed by the AXI4-Lite interface. Average fidelity was $\sim$96–97%, sufficient to demonstrate Fourier interference patterns but short of research-grade accuracy. Importantly, the errors scale with circuit depth, suggesting that larger-qubit emulation would require either wider datapaths or floating-point support.

### *C. Visualisation and Usability*

The interactive GUI, integrating PYNQ with Jupyter and Voila, was essential in bridging numerical results with intuitive quantum state understanding. Real-time Bloch sphere visualisation confirmed expected transformations and was particularly valuable in identifying phase-related discrepancies that raw amplitude values alone did not reveal. This strengthens the system's role as an educational platform, where accessibility and interpretability are as critical as raw numerical fidelity.

### *D. Overall Assessment*

The study demonstrates that SoC-based emulation offers a feasible hybrid approach: sufficiently accurate for teaching and small-scale research, while constrained by fixed-point arithmetic and interface bandwidth. The work bridges the gap between software simulators and costly quantum hardware by enabling hands-on experimentation on accessible embedded platforms. Future improvements, such as higher-precision arithmetic, optimised scaling, and refined GUI interaction, could further enhance the balance between accuracy, usability, and resource efficiency.

## VIII. Conclusion

This work demonstrated the feasibility of emulating quantum gates and small circuits on a Zynq UltraScale+ MPSoC platform. By leveraging the programmable logic (PL) for gate-level arithmetic and the processing system (PS) for software integration, the emulator achieved accurate reproduction of single- and two-qubit gates, as well as a three-qubit Quantum Fourier Transform (QFT). Execution times in the millisecond range highlight the efficiency of pipelined hardware implementations compared to conventional software simulators.

While small numerical deviations were observed—primarily due to fixed-point precision, AXI transfer width limitations, and truncation during arithmetic—the overall behaviour of each gate and the QFT was preserved. Importantly, integration with a Python-based interface and Bloch sphere visualisations ensured that the system not only reproduced results numerically but also conveyed quantum concepts in an intuitive and educational manner.

These findings establish SoC-based emulation as a practical middle ground between software simulation and dedicated quantum processors, making quantum computing concepts more accessible to both researchers and students.

## IX. Future Work

Several directions can extend this work. Increasing datapath precision (e.g., moving from 8- or 16-bit to wider fixed-point or floating-point representations) would reduce phase and amplitude errors, particularly for superposition-heavy circuits such as the Hadamard and QFT. Extending the design to support more qubits would enable demonstration of larger algorithms such as Shor's factorisation, though this will require optimised resource allocation and careful scaling of arithmetic units.

On the usability side, enhancing the graphical interface to support richer interactive visualisation would further improve accessibility for teaching. Combining these improvements would strengthen the role of MPSoC-based emulators as a bridge technology—supporting education, prototyping, and hybrid quantum–classical workflows while quantum hardware matures.

## References

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge, UK: Cambridge University Press, 2000.

[2] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.

[3] Qiskit: An Open-Source Quantum Computing Framework, IBM Quantum, 2024. [Online]. Available: https://www.ibm.com/quantum/qiskit

[4] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Improving gate-level simulation of quantum circuits," *Quantum Inf. Process.*, vol. 2, pp. 347–380, 2003.

[5] E. Gutierrez et al., "Quantum computer simulation using the CUDA programming model," *Comput. Phys. Commun.*, vol. 181, no. 2, pp. 283–300, 2010.

[6] M. Khalid, U. Mujahid, A. Jafri, H. Choi, and N. u. I. Muhammad, "An FPGA-based hardware abstraction of quantum computing systems," *J. Comput. Electron.*, vol. 20, no. 5, pp. 2001–2018, 2021, doi: 10.1007/s10825-021-01765-w.

[7] M. Aminian et al., "FPGA-based circuit model emulation of quantum algorithms," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2008, pp. 399–404.

[8] M. A. L. Y. H. a. M. M. N. Khalil-Hani, "An Accurate FPGA-Based Hardware Emulation on Quantum Fourier Transform," vol. 163, pp. 23–30, 2015.

[9] Y. H. Lee, M. Khalil-Hani, and M. N. Marsono, "An FPGA-Based Quantum Computing Emulation Framework Based on Serial-Parallel Architecture," *Int. J. Reconfig. Comput.*, vol. 2016, 2016, doi: 10.1155/2016/5718124.

[10] Y. Lee, M. Khalil-Hani, and M. N. Marsono, "An FPGA-based quantum circuit emulation framework using Heisenberg representation," *Int. J. Quantum Inf.*, vol. 16, p. 1850052, 2018, doi: 10.1142/S0219749918500521.

[11] J. D. J. Pilch, "An FPGA-based real quantum computer emulator," *J. Comput. Electron.*, vol. 18, 2019, doi: 10.1007/s10825-018-1287-5.

[12] Xilinx, "PYNQ: Python Productivity for Zynq," [Online]. Available: https://pynq.io

[13] L. Crockett, D. Northcote, C. Ramsay, F. D. Robinson, and R. W. Stewart, ”Exploring Zynq MPSoC With PYNQ and Machine Learning Applications,” Strathclyde Academic Media, 2019.