Towards Complete and Scalable Emulation of Quantum Algorithms on High-Performance Reconfigurable Computers

Esam El-Araby[†], Naveed Mahmud[††], Mingyoung Joshua Jeng[†], Andrew MacGillivray[†], Manu Chaudhary[†], Md. Alvir Islam Nobel[†], SM Ishraq Ul Islam[†], David Levy[†], Dylan Kneidel[†], Madeline R. Watson[†], Jack G. Bauer[†], and Andrew E. Riachi[†]
[†] *Electrical Engineering and Computer Science,* University of Kansas, Lawrence, KS, USA
[†]{esam, mingyoungjeng, amacgillivray, manu.chaudhary, islam.alvir, ishraq, david.levy, dckneidel, madeline.watson, jack.bauer, and ariachi}@ku.edu
[††] *Computer Engineering and Sciences,* Florida Institute of Technology, Melbourne, FL, USA
[††]nmahmud@fit.edu

Abstract—Contemporary quantum computers face many critical challenges that limit their usefulness for practical applications. A primary limiting factor is classical-to-quantum (C2Q) data encoding, which requires specific circuits for quantum state initialization. The required state initialization circuits are often complex and violate decoherence constraints, particularly for I/O intensive applications. Existing Noisy Intermediate-Scale Quantum (NISQ) devices are noise-sensitive and have low quantum bit (qubit) counts, thus limiting the applicability of C2Q circuits for encoding large and realistic datasets. This has made the study of complete and realistic circuits that include data encoding challenging and has also led to a heavy dependency on costly and resource-intensive simulations on classical platforms. In this work, we propose a cost-effective, classical-hardware-accelerated framework for realistic and complete emulation of quantum algorithms. The emulation framework incorporates components for the critical C2Q data encoding process, as well as architectures for quantum algorithms such as the quantum Haar transform (QHT). The framework is used to investigate optimizations for C2Q and QHT algorithms, and the corresponding optimized quantum circuits are presented. The framework is implemented on a High-Performance Reconfigurable Computer (HPRC) which emulates the proposed QHT circuits combined with proposed C2Q data encoding methods. For performance benchmarks, CPU-based emulations and simulations on a state-of-the-art quantum computing simulator are also carried out. Results show that the proposed hardware-accelerated emulation framework is more efficient in terms of speed and scalability compared to CPU-based emulation and simulation.

Index Terms—Quantum Computing, Quantum Encoding, Field-Programmable Gate Arrays, Parallel Processing

1 Introduction

Quantum computing offers potential advantages over classical computing in certain problem domains, such as the bounded-error quantum polynomial (BQP) class of decision problems [1] [2]. BQP is the class of problems that a quantum computer can solve in polynomial time, e.g., computation of discrete logarithms, factorization of composite integers, and estimation of eigenvalues [1] [3]. The potential applications for such problem-solving capabilities has lead to a growing focus on quantum technology and the development of quantum devices by a number of major international corporations as well as new start-ups [4]–[8].

State-of-the-art quantum processors are currently represented by Noisy Intermediate-Scale Quantum (NISQ) devices [9], which have several challenges that limit their usefulness. A critical challenge for NISQ devices is decoherence, a process in which the state of the quantum computer is destroyed by interaction with the environment [10]. Decoherence places constraints on the realistic applicability of quantum algorithms, since real-life applications usually require complex equivalent quantum circuits to be realized. The scale of NISQ devices also creates compounding spatial and temporal limitations. With few fully-connected quantum bits (qubits), circuit implementations must be *deep* rather than *wide* [11]. Quantum circuits with large *depth* [11], i.e., the number of time steps the circuit executes, take longer time to run and exacerbate the impact of qubit noise and decoherence. More specifically, the accumulation of errors due to noise and circuit complexity could result in incorrect results [12]. These limitations make practical implementations of quantum algorithms exceedingly challenging. Therefore, circuit optimization techniques for reducing circuit depth are needed to mitigate the effects of decoherence when using current quantum hardware.

Another critical challenge that limits the use of quantum computers in I/O-intensive, real-world applications is encoding classical data onto the quantum computer. Before executing a quantum algorithm, a quantum system typically starts from a ’ground’ or ’zero’ state [13]. In addition to the equivalent circuit of the quantum algorithm, a *state synthesis* circuit is required for initializing the quantum state with input data. The additional state synthesis circuit encodes classical data into an equivalent representation in the quantum domain, a process we have termed in this work as classical-to-quantum (C2Q) data encoding. The C2Q process adds to the algorithmic spatial and temporal complexity, and it is critical to reduce the depth of the equivalent state synthesis circuit.

To investigate the challenges associated with modern quantum devices, the use of classical computing to emulate quantum algorithms has become increasingly common. Efficient emulation allows for faster experimentation and prototyping of quantum circuits, algorithms, and applications. However, classical emulation of quantum computation generally involves the use of large-scale, resource-hungry, and costly simulation platforms. Moreover, many simulation tools do not take advantage of hardware acceleration [14]. In this regard, more efficient, scalable, and cost-effective tools for hardware-accelerated emulation of quantum algorithms are required. The use of Field-Programmable Gate Arrays (FPGAs) to emulate quantum computation has been increasing in recent times [15] due to their cost-effectiveness, high-performance, and reconfigurability.

In this paper, we demonstrate a *complete*, FPGA-accelerated, hardware framework for noise-free emulation of quantum algorithms. We refer to the term ”complete” to describe emulation that incorporates methods for quantum state initialization in addition to emulating quantum algorithms, rather than assuming a pre-initialized state. As a case study, we investigate a quantum image processing application and evaluate it using the proposed emulation framework. The application consists of encoding a classical image onto a quantum circuit and applying a Quantum Haar transform (QHT) [16] [17]. We propose quantum circuits for performing C2Q data encoding and performing a multi-dimensional, multi-level decomposable QHT. Circuit optimizations as well as detailed analysis of circuit depths for the C2Q and QHT circuits are also presented. The circuits are emulated and evaluated experimentally using the proposed emulation framework.

The hardware architectures of the emulation framework were implemented with 32-bit floating-point data precision for high accuracy and were also fully pipelined for highest throughput. For experimental evaluation, an FPGA-based, high-performance reconfigurable computing (HPRC) platform was used. RGB colored images were used as input data for the quantum image processing application. We present results for the hardware emulation runtime and resource utilization along with the corresponding analysis for the proposed emulation architectures. For verification and benchmarking the performance of the proposed emulation framework, software implementations were also performed on a CPU. In addition, the proposed quantum circuits were also implemented on a state-of-the-art quantum computing simulator. Finally, we present a quantitative comparison of our emulator with existing FPGA-based emulators in terms of scale, accuracy, and throughput. To the best of our knowledge, this work is the first FPGA-based emulation framework that integrates quantum state initialization with quantum algorithms and demonstrates the complete emulation of a full quantum image processing application.

The rest of the paper is organized as follows. Section 2 discusses background concepts and existing work. Section 3 contains the proposed quantum circuits. Section 4 contains the hardware architectures for emulation in addition to the experimental work and results. Finally, Section 5 contains the conclusion and discussion of future work.

2 Background and Related Work

This section outlines relevant prerequisite information for classical-to-quantum (C2Q) data encoding and quantum Haar transform (QHT). Related work involving data encoding, quantum wavelet/Haar transforms, and GPU/FPGA emulation is also presented.

2.1 Qubit Superposition

A quantum bit (qubit) is the basic unit of information in a quantum computer [18]. A qubit exists in a superposition of two orthogonal basis states $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$, see (1). The pure state of a qubit is represented by a normalized vector $|\psi\rangle$ such that $\langle\psi|\psi\rangle = \sqrt{|\alpha|^2 + |\beta|^2} = 1$, where α and β are complex coefficients, and $|\alpha|^2$ and $|\beta|^2$ are the probabilities of finding the qubit in the basis states $|0\rangle$ and $|1\rangle$, respectively. The state of a qubit can also be pictured as being a point on the surface of a unit sphere called the Bloch sphere [19].

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \equiv \begin{bmatrix}\alpha\\ \beta\end{bmatrix} \tag{1}$$

For generality purposes that we will use later, an arbitrary (non-normalized) two-dimensional complex vector $|\psi\rangle$ could also be pictured as being a point on the surface of a non-unit Bloch sphere of radius $r \neq 1$ as described by (2) [18] [20].

$$|\psi\rangle = re^{i\frac{t}{2}}\left[e^{-i\frac{\phi}{2}}\cos\frac{\theta}{2}\,|0\rangle + e^{i\frac{\phi}{2}}\sin\frac{\theta}{2}\,|1\rangle\right] \tag{2}$$

The angles θ and ϕ are called the elevation and azimuth angles, respectively, where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$. The angle t is usually referred to as the global phase, which is physically unobservable [18] [20]. The factor r is typically equal to unity for qubit normalized states, such that $\langle\psi|\psi\rangle = 1$ and the probability interpretation is applicable. However, similar to the global phase t, we will interpret and refer to the factor r as the $global\ scale$ and picture it as the radius of a non-unit Bloch sphere.

2.2 Decoherence

Environmental effects can alter the coherence of a qubit's state in the form of the so-called decoherence noise [10]. Interaction with the environment erodes the qubit's state to become gradually mixed, which leads to information loss [10] [18]. As time progresses, quantum interference is suppressed and the ability to perform additional operations is lost [10] [18]. Decoherence time creates a real-time constraint for quantum circuits, such that circuits must be optimized so that the output can be captured within the decoherence time. This constraint is typically considered in the form of relaxation (T1) or dephasing (T2) times, which represent how long for a qubit in a given state to either return to its ground state or succumb to environmental noise, respectively [10].

2.3 Quantum Gates

In the circuit model of quantum computing [18], computation begins with the system in an unentangled quantum state $|\psi_0\rangle = |0\rangle^{\otimes n}$, where n is the number of qubits, which can be expressed as a superposition of $N = 2^n$ basis states. Depending on the quantum algorithm, different unitary transformations or quantum gates can be applied to reach a final quantum state $|\psi\rangle$. In this work, we utilize the Hadamard (H), SWAP, Controlled-NOT (CNOT), Rotation (R_y, R_z), Rotate-Left (RoL) and Rotate-Right (RoR) gates [18] [21], see Fig. A1 (in the Appendix). We also denote the time delays of the H and the SWAP gates as τ_H and τ_{SWAP}, respectively.

2.4 Quantum Haar Transform (QHT)

The classical wavelet transform (WT) uses non-sinusoidal functions called mother wavelets to decompose signals/data into its spatio-temporal spectral components [22]. Due to high computational efficiency, WT is commonly used for image processing applications [22]. The first and simplest wavelet is the Haar wavelet. The Haar mother wavelet function can be constructed using a unit step function $u(t)$, as shown in (3), where a and b are the time dilation and displacement factors, respectively. The discretized version of the Haar wavelet function is defined in (4), and the expression for discrete Haar Transform on a signal $f(t)$ is given by (5), where $t = q \cdot \Delta t$, $b = j \cdot \Delta t$, $a = K \cdot \Delta t$, Δt is the sampling period, and K is the Haar window size in samples.

$$\Psi\left(\frac{t-b}{a}\right) = u\left(\frac{t-b}{a}\right) - 2u\left(\frac{t-b}{a} - \frac{1}{2}\right) + u\left(\frac{t-b}{a} - 1\right) \tag{3}$$

$$\Psi_D\left(\frac{q-j}{K}\right) = \begin{cases} +1, & 0 \leq (q-j) < \frac{K}{2} \\ -1, & \frac{K}{2} \leq (q-j) < K \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$F_D(j, K) = \sum_{q=0}^{N-1} f_D(q \cdot \Delta t)\Psi_D\left(\frac{q-j}{K}\right) \tag{5}$$

The quantum equivalent of the Haar transform, i.e., the quantum Haar transform (QHT) [16] [17], is used in the quantum-information-processing (QIP) domain. Signal samples are encoded as the basis state coefficients of a quantum state $|\psi\rangle$ in superposition, as shown in (6). In QHT, the Haar Transform is applied on the coefficients, and the equivalent expression for the output of the QHT is given by (7);

$$|\psi\rangle = \sum_{q=0}^{N-1} f(q \cdot \Delta t)\,|q\rangle\,,\ \text{where} \sum_{q=0}^{N-1} |f(q \cdot \Delta t)|^2 = 1 \tag{6}$$

$$|\psi\rangle_{\text{QHT}} = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{q=0}^{N-1} f(q \cdot \Delta t)\Psi_D\left(\frac{q-j}{K}\right)|j\rangle \tag{7}$$

where K is the wavelet window size, n is the number of qubits, Δt is the sampling period, Ψ_D is the Haar mother wavelet function in complex conjugate form, $N = 2^n$ is the number of data samples represented as the total number of quantum basis-states, $|\psi\rangle$ is the input state, and $|\psi\rangle_{\text{QHT}}$ is the output state.

2.5 Quantum Permutations

The fundamental operations used in QHT, and also in many classical computations that involve signal and image processing, are permutations in general, and specifically perfect-shuffle-permutations (PSP) [16]. Quantum permutations are described by their effect on the ordering of qubits [16] [17]. For building the proposed QHT circuits, we will focus on the two PSP operations Rotate-Left (RoL) and Rotate-Right (RoR). RoL and RoR operations, given by (8) and (9), respectively, are fundamentally circular left/right shift operations on a register of n qubits. Notably, these can be implemented with a network of quantum SWAP gates [16], see Fig. A1 (in the Appendix).

$$\text{RoL}(n-1, 0) : |q_{n-1}q_{n-2}\cdots q_1q_0\rangle \mapsto |q_{n-2}\cdots q_1q_0q_{n-1}\rangle \tag{8}$$

$$\text{RoR}(n-1, 0) : |q_{n-1}q_{n-2}\cdots q_1q_0\rangle \mapsto |q_0q_{n-1}q_{n-2}\cdots q_1\rangle \tag{9}$$

2.6 Related Work

In this section, we discuss the work related to classical-to-quantum (C2Q) data encoding, quantum wavelet transforms, and hardware accelerated (GPU/FPGA-based) emulation of quantum computations.

2.6.1 C2Q Data Encoding

Existing methods of encoding classical data to a quantum representation are of three types: (a) basis encoding, (b) angle encoding, and (c) amplitude encoding [23] [24].

Basis encoding involves encoding the binary representations of data points as basis states. Typically, this technique is costly in terms of number of qubits. The authors in [25] presented an optimized basis encoding technique for image processing, where pixels are represented by the tensor product of their color and position. As a result of their optimization, the qubit cost was lowered, however their technique incurred greater circuit depth.

Angle encoding represents one data point per qubit, with each data point encoded as a normalized rotation in the Bloch sphere. The authors in [26] investigated angle encoding and presented a quantum method for image edge detection. However, their method could be impractical, because each image pixel requires one qubit for encoding.

In amplitude encoding, each data point is represented as the amplitude/coefficient of a basis state in a superimposed quantum state. The work in [27] proposed circuits with depth complexity $O(n)$, where n is the number of qubits.

However, the number of qubits required is of the order $O(N)$, where $N = 2^n$ is the data size, therefore the proposed circuits are not feasible for current or near-future quantum processors. In [28], the authors presented a quantitative analysis of existing amplitude encoding methods and proposed state synthesis circuits based on amplitude encoding with 50% reduction in circuit depth and 50% reduction of total number of gates. The authors in [20] also presented a method of arbitrary state synthesis for amplitude encoding, which was used by IBM Quantum (IBM-Q) [29] Qiskit APIs for state synthesis [30]. Among the data encoding methods, amplitude encoding requires the least number of qubits. To represent a data size of $N = 2^n$, it requires $\lceil \log_2(N) \rceil$ qubits with a depth complexity of $O(N)$. However, this technique requires the implementation of complex quantum circuits.

In this work, we extend the methods in [20] and [28] to present a more optimized method based on amplitude encoding for classical-to-quantum data encoding. The corresponding quantum circuits for our proposed methods and detailed analysis of circuit depths are also presented.

2.6.2 *Quantum Wavelet and Haar Transforms*

Fijany and Williams proposed methods for the implementation of permutation matrices for quantum wavelet transform (QWT), which is the quantum equivalent of the classical wavelet transform [16]. In [17], partial quantum circuit derivations for the Haar and Daubechies wavelet were presented. The authors also proposed implementation circuits of multi-level and multi-dimensional packet QWT. In general, the previously reported work on QWT and QHT have not investigated data initialization methods, circuit optimizations, nor actual hardware implementations. In our previous investigations on QHT [31], we proposed optimizations to reduce circuit depth, and in this work we leverage and extend the work in [31] with integration of data initialization methods and implementations on classical hardware.

2.6.3 *GPU-based Emulation*

In our survey of related quantum emulators, we focused on the recently reported hardware-accelerated quantum simulators, e.g., GPU-based quantum simulators/emulators. There is a plethora of open-source GPU-based quantum simulators/emulators [32] that have been introduced in recent years. The major constraint of these simulators is their heavy dependency on the availability of large memory and costly high-end GPUs in their local hosts. In [33], the authors introduced an open-source quantum numerical emulator with single and double precision floating point formats that offers larger circuit simulations using costly high-end GPU-accelerated systems. This simulator can simulate only up to 29-qubit circuits using a 12 GB GPU system, and up to 38-qubit circuits on a high-performance supercomputer with 8 TiB of memory. In [34], the authors proposed a GPU-accelerated quantum simulation framework available in both single and double precision that only allows 29-qubit circuit simulations using high-end GPUs with 16 GB memory. In addition, there are several high-performance quantum simulators available, such as [29] or [35], that can execute complex circuit simulations with higher qubit count. However, the communication overhead among simulation servers is not negligible and incurs a higher time overhead that adds to the total execution time.

2.6.4 *FPGA-based Emulation*

Reported FPGA emulation of quantum algorithms in [36]–[42] are generally characterized by low scalability (small number of qubits), low accuracy (fixed-point precision), and low throughput (low operating frequency). In addition, the related work assumes that quantum data is already initialized and there is no integration of classical-to-quantum (C2Q) data encoding methods, which makes the emulation unrealistic and thus incomplete.

In this work, we propose a cost-effective, FPGA-accelerated methodology for *complete* emulation of quantum computation. Our methodology is the first to integrate amplitude-encoding-based quantum state initialization methods with algorithm emulation. We present two methods and their corresponding quantum circuits for classical-to-quantum (C2Q) data encoding/state initialization which have less depth and complexity compared to existing methods. We also propose optimizations for the quantum Haar transform (QHT) to reduce circuit depth. Finally, the integration of C2Q data encoding with QHT is evaluated experimentally by emulation.

3 Proposed Quantum Circuits

In this section, we present quantum circuits for classical-to-quantum (C2Q) data encoding and the quantum Haar transform (QHT). Further optimizations for the C2Q and QHT circuits are also presented.

3.1 Classical-to-Quantum (C2Q) Data Encoding

The process of synthesizing and initializing a quantum state with arbitrary classical data is termed as classical-to-quantum (C2Q) data encoding in this work. We propose two methods and corresponding quantum circuits for C2Q data encoding. In the first proposed method (Method 1), we use non-unitary operators and non-normalized state vectors where the global scale $r \neq 1$. This method is suitable only for emulation purposes. In the second proposed method (Method 2), we use unitary operators and normalized state vectors where the global scale $r = 1$, which makes it suitable for physical realization and implementation. Given a classical dataset of $N = 2^n$ elements, where n is the number of required qubits to represent the classical dataset, we propose a quantum circuit denoted as U^{C2Q-1} for Method 1 that synthesizes a corresponding quantum state with encoded classical data. U^{C2Q-1} is parameterized by the global scale $r \neq 1$, global phase t, elevation angle θ, and azimuth angle ϕ. We also present an optimized state synthesis circuit U^{C2Q-2} for Method 2 that is characterized by unity global scale, i.e., $r = 1$. The steps of the proposed methodology in the formation of the circuits U^{C2Q-1} and U^{C2Q-2} are elaborated in the next subsections.

3.1.1 *Method 1: Non-Unitary State Synthesis for Emulation*

A quantum register of n qubits in ground state is denoted by $|\psi_0\rangle = |0\rangle^{\otimes n}$. In quantum amplitude encoding, given a classical data set of $N = 2^n$ elements, we want to synthesize a quantum state $|\psi\rangle$, see (10), and encode each data element as a basis state coefficient, α_i.

$$|\psi\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle, \text{ where } \sum_{i=0}^{N-1} |\alpha_i|^2 = 1 \tag{10}$$

For non-unitary state synthesis, it is required to find a non-unitary operator, U^{C2Q-1}, that transforms $|\psi_0\rangle$ to $|\psi\rangle$. Any arbitrary single-qubit non-unitary gate can be decomposed as a series of R_z and R_y gates known as the ZYZ or Pauli decomposition [18], see (11).

$$\begin{aligned} |\psi\rangle &= R_z(\phi) \cdot R_y(\theta) \cdot re^{i\frac{t}{2}} \cdot |0\rangle \\ &= R_z(\phi) \cdot R_y(\theta) \cdot R_z(-t) \cdot r \cdot |0\rangle \end{aligned} \tag{11}$$

Therefore, a qubit in ground state $|0\rangle$ can be transformed to any arbitrary non-normalized vector $|\psi\rangle$ by applying a Bloch sphere radius scaling r, followed by a $-t$ rotation about the z-axis, then a θ rotation about the y-axis, and finally a ϕ rotation about the z-axis. The factor $re^{i\frac{t}{2}}$ is the physically unobservable constant factor for an arbitrary non-normalized state, t is the global phase, and $r \neq 1$ is the global scale [18] [20]. Given the coefficients α and β of the target arbitrary non-normalized state $|\psi\rangle$, see (1), and using the transformation matrices of R_z and R_y, see Fig. A1 (in the Appendix), the parameters r, t, θ, and ϕ in (11) could be determined by (12).

$$\begin{aligned} &r = \sqrt{|\alpha|^2 + |\beta|^2}\ , \ t = \angle\beta + \angle\alpha \\ &\theta = 2\tan^{-1}\left(\frac{|\beta|}{|\alpha|}\right)\ , \ \phi = \angle\beta - \angle\alpha \end{aligned} \tag{12}$$

where

$$|\alpha| = \sqrt{\text{Re}^2(\alpha) + \text{Im}^2(\alpha)},\ \angle\alpha = \cos^{-1}\left(\frac{\text{Re}(\alpha)}{|\alpha|}\right),$$
$$|\beta| = \sqrt{\text{Re}^2(\beta) + \text{Im}^2(\beta)},\ \angle\beta = \cos^{-1}\left(\frac{\text{Re}(\beta)}{|\beta|}\right)$$

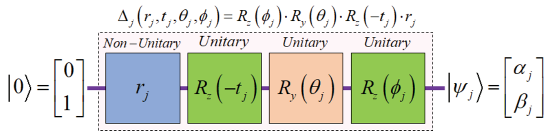

$\Delta_j(r_j,t_j,\theta_j,\phi_j) = R_z(\phi_j)\cdot R_y(\theta_j)\cdot R_z(-t_j)\cdot r_j$
Non – Unitary
Unitary
Unitary
Unitary
$|0\rangle = \begin{bmatrix}0\\1\end{bmatrix}$
r_j
$R_z(-t_j)$
$R_y(\theta_j)$
$R_z(\phi_j)$
$|\psi_j\rangle = \begin{bmatrix}\alpha_j\\\beta_j\end{bmatrix}$

Fig. 1: Pauli (ZYZ) decomposition for non-unitary arbitrary state synthesis.

Using the Pauli decomposition described by (11) and the parameters obtained by (12), we derive a method for transforming an n-qubit register in the ground state $|\psi_0\rangle = |0\rangle^{\otimes n}$ to an arbitrary state $|\psi\rangle$, see Fig. 1. To synthesize the j^{th} pair of coefficients, or $|\psi_j\rangle$ in the state vector of $|\psi\rangle$, we apply Δ_j on a ground state $|0\rangle$, where $j = 0, 1, 2, \cdots, (2^{n-1} - 1)$. Note that $|\psi_j\rangle$ is generally non-normalized, and thus each Δ_j is non-unitary, see Fig. 1. However, Δ_j cannot be applied on one qubit in the n-qubit register without also affecting the other coefficients in $|\psi\rangle$. Hence, each transformation Δ_j needs to be applied conditionally to synthesize the j^{th} pair of coefficients in the output state. The resulting conditional quantum circuit can be represented by a block-diagonal matrix U_{block} of which each diagonal block is a 2×2 transformation matrix Δ_j, see Fig. 1 and (13). The elements of Δ_j are calculated using the parameters r_j, t_j, θ_j, and ϕ_j obtained from the j^{th} pair of coefficients using (12).

$$\begin{aligned} U_{block} &= \Delta_0 \oplus \Delta_1 \oplus \cdots \Delta_j \cdots \oplus \Delta_{(2^{n-1}-1)} \\ &= \text{diag}(\Delta_0, \Delta_1, \cdots, \Delta_j, \cdots, \Delta_{(2^{n-1}-1)}) \end{aligned} \tag{13}$$

A block-diagonal matrix such as U_{block} can be implemented as a uniformly-controlled circuit or a quantum multiplexer [20]. A uniformly-controlled circuit is one where a different operation or gate is applied to the target qubit (least significant qubit in this case), for all possible combinations of the remaining control qubits. The uniformly-controlled circuit corresponding to U_{block}, see Fig. A2a (in the Appendix), contains n qubits of which $(n-1)$ are control qubits controlling the operation on the least significant target qubit. For each combination of the control qubits, the corresponding Δ_j is applied on the target qubit, where $j = 0, 1, 2, \cdots, (2^{n-1}-1)$. To produce all combinations on the control qubits with equal probability, a set of H gates must be applied on the $(n-1)$ control qubits before applying the U_{block} transformation. The desired final state $|\psi\rangle$ is produced at the output with the target coefficients as a result of uniformly applying each Δ_j transformation on the least significant qubit. The overall transformation, U^{C2Q-1}, from ground state $|\psi_0\rangle = |0\rangle^{\otimes n}$ to $|\psi\rangle$ can be expressed by (14), see also Fig. A2 (in the Appendix).

$$\begin{aligned} |\psi\rangle &= U^{C2Q-1} \cdot |\psi_0\rangle = U^{C2Q-1} \cdot |0\rangle^{\otimes n}, \text{ where} \\ U^{C2Q-1} &= U_{block}(r,t,\theta,\phi) \cdot \left(H^{\otimes(n-1)} \otimes I\right) \\ &= U_0^{C2Q-1}(t,\theta,\phi) \cdot U_{rem}^{C2Q-1}(r) \end{aligned} \tag{14}$$

Each Δ_j block is a sequence of global scale, followed by z-rotation, followed by y-rotation, and followed by z-rotation as shown in Fig. A2b (in the Appendix), and Δ_j is calculated from the corresponding set of parameters $\{r_j \cdot 2^{(\frac{n-1}{2})}, t_j, \theta_j, \phi_j\}$ obtained by (12). Note that the global scale r_j has been modified by a factor of $2^{(\frac{n-1}{2})}$ resulting from the application of the H gates on the $(n-1)$ control qubits. Since each set of operations is mutually exclusive from the others, we can separate them into uniformly-controlled groups of scale, z-rotations, y-rotations, and z-rotations as shown in Fig. A2c (in the Appendix). To represent uniformly-controlled operations as a single gate operation, we use the 'square box' notation [20] for the control bits, and the parameterized operations on the data qubit are replaced by a single box denoting the operation. We use this notation to simplify the circuit in Fig. A2c (in the Appendix) and the resulting circuit representation is shown in Fig. 2.

It is worth mentioning that the $U_{rem}^{C2Q-1}(r)$ operator, shown in (14) and Fig. 2, is non-unitary. For this reason, Method 1 is only suitable for emulation purposes rather than for physical realization and implementation on actual quantum devices which require all operators to be unitary. Therefore, we are next presenting our proposed Method 2 for unitary state synthesis.

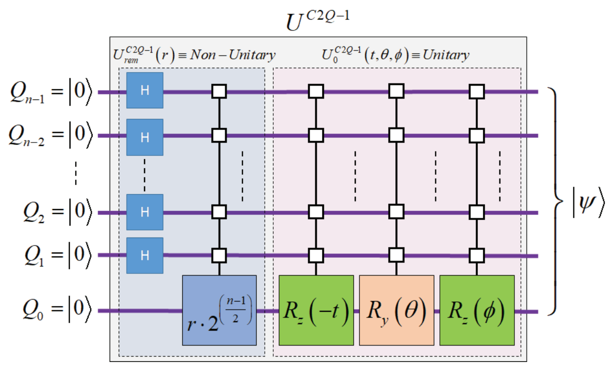


Fig. 2: Simplified full quantum circuit for non-unitary C2Q (Method 1) data encoding with uniformly-controlled operations.

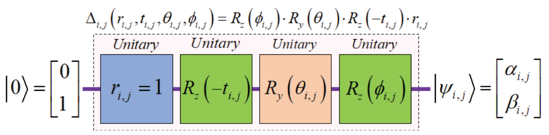
3.1.2 Method 2: Unitary State Synthesis for Physical Realization

Here, we propose a more practical approach for synthesizing a quantum state in which the global scale is unity, i.e., $r = 1$, see Fig. 3a. As a result, all operators in this circuit are unitary and are physically implementable. In developing our Method 2, we leverage and improve the recursive approach discussed in [20] which is reported to be the most efficient technique for state synthesis [20] and hence it is used by IBM-Q Qiskit [30]. We present the corresponding quantum circuit U^{C2Q-2} consisting of a pyramidal structure of U_j gates, where $j = 0, 1, \cdots, (n-1)$, see Fig. 3b and Fig. A3a (in the Appendix), to eliminate the non-unitary global scale term of Method 1. Fig. 3c and (15) show and describe the method proposed by Shende *et al.* [20]. For optimizing the circuit depth and eliminating the residual global phase term $e^{i\frac{t_{n-1}}{2}}$, see Fig. 3c and (15), of Shende's method [20], we moved all the R_z rotation operators to the first step U_0 of the process, where a set of $R_z(-t)$, $R_y(\theta)$, and $R_z(\phi)$ rotations are used in U_0, see Fig. 3d. A set of only $R_y(\theta)$ rotations is then applied in the remaining $(n-1)$ operations U_j, where $j = (n-1), (n-2), \cdots, 2, 1$, see Fig. 3d and (16).

$$\begin{aligned}
|\psi\rangle &= U^{Shende} \cdot |\psi_0\rangle = U^{Shende} \cdot |0\rangle^{\otimes n}, \text{ where} \\
U^{Shende} &= \left(\prod_{j=0}^{n-1} U_j^{Shende}(\theta,\phi) \otimes I^{\otimes j}\right) \cdot e^{i\frac{t_{n-1}}{2}} \\
&= U_0^{Shende}(\theta,\phi) \cdot U_{rem}^{Shende}(\theta,\phi) \cdot e^{i\frac{t_{n-1}}{2}}
\end{aligned} \tag{15}$$

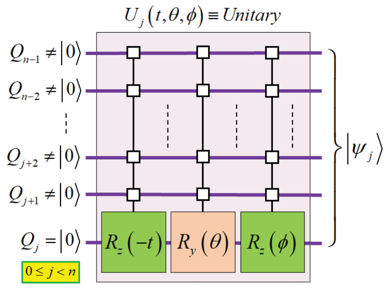
$$\begin{aligned}
|\psi\rangle &= U^{C2Q-2} \cdot |\psi_0\rangle = U^{C2Q-2} \cdot |0\rangle^{\otimes n}, \text{ where} \\
U^{C2Q-2} &= U_0^{C2Q-2}(t,\theta,\phi) \cdot \left(\prod_{j=1}^{n-1} U_j^{C2Q-2}(\theta) \otimes I^{\otimes j}\right) \\
&= U_0^{C2Q-2}(t,\theta,\phi) \cdot U_{rem}^{C2Q-2}(\theta)
\end{aligned} \tag{16}$$

Each U_j is a uniformly-controlled operation, shown in Fig. 3b. As shown in Fig. A3a (in the Appendix), a number ($k_j = 2^{(n-1-j)}$) of $\Delta_{i,j}$ rotation operations are applied for each U_j, where $0 \le i < k_j$, and $0 \le j < n$. Each $\Delta_{i,j}$ operation requires calculating a 2-tuple parameters $(\alpha_{i,j}, \beta_{i,j})$ from given classical data set $|\psi\rangle$ using the steps described in

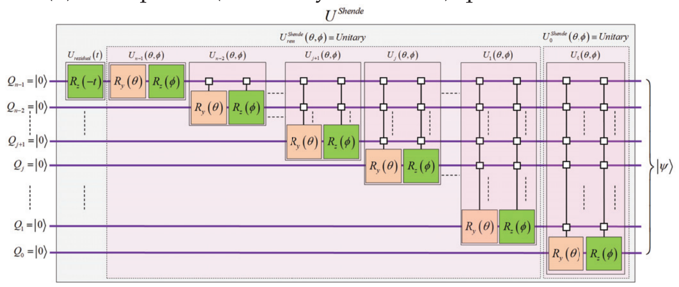


$\Delta_{i,j}(r_{i,j}, t_{i,j}, \theta_{i,j}, \phi_{i,j}) = R_z(\phi_{i,j}) \cdot R_y(\theta_{i,j}) \cdot R_z(-t_{i,j}) \cdot r_{i,j}$
Unitary
Unitary
Unitary
Unitary
$|0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
$r_{i,j} = 1$
$R_z(-t_{i,j})$
$R_y(\theta_{i,j})$
$R_z(\phi_{i,j})$
$|\psi_{i,j}\rangle = \begin{bmatrix} \alpha_{i,j} \\ \beta_{i,j} \end{bmatrix}$

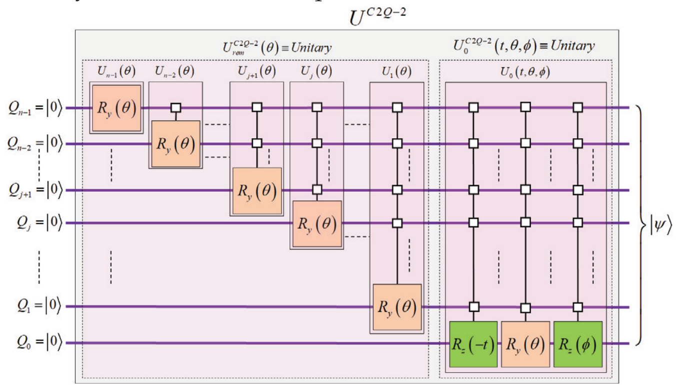
(a) Pauli (ZYZ) decomposition for single-qubit state synthesis.



(b) Multiplexer (uniformly-controlled) quantum circuit.



(c) Pyramidal structure of quantum circuit for Shende [20].



(d) Pyramidal structure of quantum circuit for Method 2.

Fig. 3: Quantum circuits for unitary C2Q data encoding for Shende [20] and proposed Method 2.

equations (17) to (20) which are then used in (12) to calculate the required 4-tuple parameters $(r_{i,j}, t_{i,j}, \theta_{i,j}, \phi_{i,j})$. It could be seen from (17) to (20) that unitarity of transformation is maintained where $r_{i,j} = \sqrt{|\alpha_{i,j}|^2 + |\beta_{i,j}|^2} = 1$, see Fig. 3a.

$$|\psi\rangle = \begin{bmatrix} C_0 \\ C_1 \\ .. \\ .. \\ C_{N-1} \end{bmatrix}, \; N = 2^n, \; n \equiv \text{number of qubits} \tag{17}$$

$$P_{i,j} = \begin{cases} |C_{2i}|^2 + |C_{2i+1}|^2, & j = 0, \; 0 \le i < 2^{(n-1)} \\ P_{2i,j-1} + P_{2i+1,j-1}, & 1 \le j < n, \; 0 \le i < 2^{(n-1-j)} \\ 0, & 2^{(n-1-j)} \le i < 2^{(n-1)} \end{cases} \tag{18}$$

$$\alpha_{i,j} = \begin{cases} \frac{C_{2i}}{\sqrt{P_{i,j}}}, & P_{i,j} \neq 0,\ j = 0,\ 0 \leq i < 2^{(n-1)} \\ \sqrt{\frac{P_{2i,j-1}}{P_{i,j}}}, & P_{i,j} \neq 0,\ 1 \leq j < n,\ 0 \leq i < 2^{(n-1-j)} \\ 1, & P_{i,j} = 0 \end{cases} \tag{19}$$

$$\beta_{i,j} = \begin{cases} \frac{C_{2i+1}}{\sqrt{P_{i,j}}}, & P_{i,j} \neq 0,\ j = 0,\ 0 \leq i < 2^{(n-1)} \\ \sqrt{\frac{P_{2i+1,j-1}}{P_{i,j}}}, & P_{i,j} \neq 0,\ 1 \leq j < n,\ 0 \leq i < 2^{(n-1-j)} \\ 0, & P_{i,j} = 0 \end{cases} \tag{20}$$

where $0 \leq j < n$, $0 \leq i < k_j$, and $k_j = 2^{(n-1-j)}$.

3.1.3 Analysis of Circuit Depth

We determined the circuit depths for our proposed C2Q Method 2 in comparison with the most efficient reported method by Shende *et al.* [20]. We followed a similar approach to Shende's in [20] in which we considered counting the number of 1-qubit rotation gates and 2-qubit CNOT gates. We focused on the gate count of the 2-qubit CNOT gates since they have significant impact on the overall circuit fidelity due to their higher gate errors compared to single-qubit gates. In addition, we distinguished between two types of input data: (a) complex data and (b) positive real data. The derived circuit depths for each method are summarized in Table 1. Although C2Q Method 1 is targeted for emulation and is not implementable on physical devices, we've included it in our circuit depth analysis for completeness of analysis. It will also serve as a basis on which we could theoretically and experimentally compare our different implementations, i.e., on FPGAs, CPUs, and IBM Quantum (IBM-Q) [29] Qiskit cloud-based simulator.

In our analysis, each n-qubit uniformly-controlled R_y and R_z rotation operation in Figs. 2 and 3 can be decomposed into a sequence of 2^n gates in total (2^{n-1} 1-qubit rotation gates + 2^{n-1} 2-qubit CNOT gates, where $n > 1$) [20] , see Fig. 4. For positive real data, i.e., $\angle\alpha = \angle\beta = 0$, all $R_z(-t)$ and $R_z(\phi)$ gates could be eliminated since $t = \phi = 0$, see (12), which results in both our proposed Method 2 and Shende's [20] being an identical pyramidal sequence of n uniformly-controlled $R_y(\theta)$ gates, see Figs. 3c, 3d, and Table 1. The advantage of our proposed Method 2 over Shende's [20] becomes obvious for complex data where the improvement/reduction of the total number of gates as well as the overall circuit depth asymptotically approaches 25% as the number of qubits increases indefinitely, see Table 1. Moreover, our proposed Method 2 for encoding complex data compared to Shende's [20] asymptotically achieves 50% reduction in the number of 2-qubit CNOT gates, see Table 1. Our analysis has been experimentally verified using IBM-Q Qiskit APIs for state synthesis [30]. Our theoretical expectations were in perfect match with the experimental measurements as it will be shown in the Experimental Results section.

3.2 Optimized Circuits for Multi-level Decomposable, Multi-dimensional Quantum-Haar-Transform (QHT)

Similar to classical wavelet transforms, d-dimensional QHT is decomposable for l levels of decomposition using either packet or pyramidal decomposition techniques [21]. For our implementation, we use lossless packet decomposition where data dimensions are not lost during decomposition. In packet decomposition, all the data qubits are needed for the entire process. The maximum number of decomposition levels l_{max}^{pkt} is dependent on the number of data dimensions d and the total number of qubits n. However, the maximum number of levels for lossless decomposition is equal to the minimum number of qubits across all d dimensions.

In packet decomposition, a d-dimensional QHT operation, $U^{d-D-QHT}$ is applied repeatedly for every level on all the data (qubits), see Fig. A4 (in the Appendix). The $U^{d-D-QHT}$ operation consists of three components: (1) input permutation operations applied to the state vector, (2) Haar transform operations, and (3) output permutations applied to produce the output state vector. Detailed algorithms

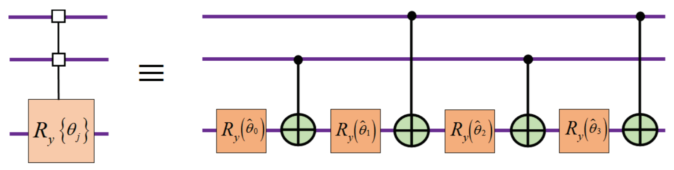


Fig. 4: Decomposition of a uniformly-controlled 3-qubit R_y rotation operation.

TABLE 1: Theoretical expectation of C2Q circuit depth and total number of gates (1-qubit rotation gates and 2-qubit CNOT gates).

			Complex Data	Positive Real Data
Reported (Shende [20])		CNOT Gates	$4 \cdot 2^{(n-1)} - 2 \cdot n - 2,\ n \geq 1$	$\begin{cases} 0, n = 1 \\ 2 \cdot 2^{(n-1)} - 2,\ n > 1 \end{cases}$
		Total Gates	$8 \cdot 2^{(n-1)} - 2 \cdot n - 3$	$4 \cdot 2^{(n-1)} - 3$
		Circuit Depth	$8 \cdot 2^{(n-1)} - 3 \cdot n - 2$	$4 \cdot 2^{(n-1)} - n - 2$
Proposed (Method 1)		CNOT Gates	$\begin{cases} 0, n = 1 \\ 2^{(n-1)},\ n > 1 \end{cases}$	$\begin{cases} 0, n = 1 \\ 2^{(n-1)},\ n > 1 \end{cases}$
		Total Gates	$\begin{cases} 3, n = 1 \\ 4 \cdot 2^{(n-1)},\ n > 1 \end{cases}$	$\begin{cases} 1, n = 1 \\ 2 \cdot 2^{(n-1)},\ n > 1 \end{cases}$
		Circuit Depth	$\begin{cases} 3, n = 1 \\ 4 \cdot 2^{(n-1)},\ n > 1 \end{cases}$	$\begin{cases} 1, n = 1 \\ 2 \cdot 2^{(n-1)},\ n > 1 \end{cases}$
Proposed (Method 2)		CNOT Gates	$2 \cdot 2^{(n-1)} - 2,\ n \geq 1$	$\begin{cases} 0, n = 1 \\ 2 \cdot 2^{(n-1)} - 2,\ n > 1 \end{cases}$
		Total Gates	$6 \cdot 2^{(n-1)} - 3$	$4 \cdot 2^{(n-1)} - 3$
		Circuit Depth	$\begin{cases} 6 \cdot 2^{(n-1)} - n - 2, 11 \leq n \leq 2 \\ 6 \cdot 2^{(n-1)} - n - 4,\ n > 2 \end{cases}$	$4 \cdot 2^{(n-1)} - n - 2$
Improvement / Reduction	Method 1 vs. Reported	CNOT Gates	$\delta(n) = \begin{cases} 0,\ n = 1 \\ \frac{3 \cdot 2^{(n-1)} - 2 \cdot n - 2}{4 \cdot 2^{(n-1)} - 2 \cdot n - 2},\ n > 1 \end{cases}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{3}{4}$	$\delta(n) = \begin{cases} 0,\ n = 1 \\ \frac{2^{(n-1)} - 2}{2 \cdot 2^{(n-1)} - 2},\ n > 1 \end{cases}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{1}{2}$
		Total Gates	$\delta(n) = \begin{cases} 0, n = 1 \\ \frac{4 \cdot 2^{(n-1)} - 2 \cdot n - 3}{8 \cdot 2^{(n-1)} - 2 \cdot n - 3},\ n > 1 \end{cases}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{1}{2}$	$\delta(n) = \begin{cases} 0, n = 1 \\ \frac{2 \cdot 2^{(n-1)} - 3}{4 \cdot 2^{(n-1)} - 3},\ n > 1 \end{cases}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{1}{2}$
		Circuit Depth	$\delta(n) = \begin{cases} 0, n = 1 \\ \frac{4 \cdot 2^{(n-1)} - 3 \cdot n - 2}{8 \cdot 2^{(n-1)} - 3 \cdot n - 2},\ n > 1 \end{cases}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{1}{2}$	$\delta(n) = \begin{cases} 0, n = 1 \\ \frac{2 \cdot 2^{(n-1)} - n - 2}{4 \cdot 2^{(n-1)} - n - 2},\ n > 1 \end{cases}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{1}{2}$
	Method 2 vs. Reported	CNOT Gates	$\delta(n) = \begin{cases} 0, n = 1 \\ \frac{2 \cdot 2^{(n-1)} - 2 \cdot n}{4 \cdot 2^{(n-1)} - 2 \cdot n - 2},\ n > 1 \end{cases}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{1}{2}$	0
		Total Gates	$\delta(n) = \frac{2 \cdot 2^{(n-1)} - 2 \cdot n}{8 \cdot 2^{(n-1)} - 2 \cdot n - 3}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{1}{4}$	
		Circuit Depth	$\delta(n) = \begin{cases} \frac{2 \cdot 2^{(n-1)} - 2 \cdot n}{8 \cdot 2^{(n-1)} - 3 \cdot n - 2},\ 1 \leq n \leq 2 \\ \frac{2 \cdot 2^{(n-1)} - 2 \cdot n + 2}{8 \cdot 2^{(n-1)} - 3 \cdot n - 2},\ n > 2 \end{cases}$ $\delta_{max} = \lim_{n\to\infty} \delta(n) = \frac{1}{4}$	

for 2D-QHT ($U^{2-D-QHT}$) and 3D-QHT ($U^{3-D-QHT}$) can be found in [31].

The operation, $U^{d-D-QHT}$, can be implemented using two types of QHT circuits: parallel (1-stage) and sequential (d-stage). For the purpose of this paper, we investigate optimization of the parallel (1-stage) QHT.

3.2.1 Unoptimized Parallel (1-stage) QHT

In QHT circuits, the Haar operation (H gates) is generally applied in parallel (1-stage), see Fig. A5 (in the Appendix), with the RoR and RoL operations grouped together into a set of preceding and proceeding permutations, respectively. The total time delay of this unoptimized circuit is derived from Fig. A5 (in the Appendix) and expressed by (21).

$$t_{\text{total}}^{\text{par, unopt, }pkt} = \left((2n - n_{(d-1)} - (2d-1)) \cdot \tau_{SWAP} + \tau_H\right) \cdot l \tag{21}$$

where n is the number of qubits, d is the number of data dimensions, τ_{SWAP} and τ_H are the time delays for SWAP and H gates, respectively, and l is the number of decomposition levels.

3.2.2 Optimized Parallel (1-stage) QHT

The parallel (1-stage) circuit of d-dimensional QHT is optimized by re-positioning the H gates separated by n_i qubits, where $0 \le i < d$, see Fig. A5 (in the Appendix). Due to this re-positioning of the H gates, no preceding permutations (RoL gates) are required. The proceeding RoR operations are also reduced in depth and can be applied in parallel as they are independent of each other. The expression for the total time delay for the optimized parallel l-level decomposable d-dimensional QHT circuits is given by

$$t_{\text{total}}^{\text{par, opt, }pkt} = ((n_{\max} - 1) \cdot \tau_{SWAP} + \tau_H) \cdot l \tag{22}$$

where n_{max} is the maximum number of qubits across all dimensions, τ_{SWAP} and τ_H are the time delays for SWAP and H gates, respectively, and l is the number of decomposition levels.

4 Experimental Results

The proposed hardware/FPGA-based emulation methodology and corresponding hardware architectures for emulation of C2Q and QHT are presented here. In our experimental work, three different implementations were carried out: (a) implementation using the reconfigurable system (CPU + FPGA), (b) implementation using only CPUs, and (c) implementation using IBM-Q Qiskit cloud-based simulator.

4.1 Hardware Architectures for Quantum Algorithm Emulation

To evaluate the proposed quantum circuits, a framework for hardware-based emulation is developed. The proposed emulation framework, shown in Fig. 5, for *complete* emulation of quantum algorithms, consists of two components: modeling C2Q data encoding and modeling the quantum algorithm computation. For C2Q data encoding, we have discussed two methods and their corresponding quantum circuits. For emulation on CPUs and FPGAs, we model both C2Q methods and their corresponding circuits, see Figs. 2 and 3. The model for quantum algorithm emulation is flexible and uses different emulation techniques [31] [43] depending on the algorithm. For example, for algorithms with dense matrices such as quantum Fourier transform and Grover's search, complex-multiply-and-accumulate (CMAC) operations [43] are employed. For algorithms with sparse transformation matrices such as QHT, we employ kernel-based operations [31]. The hardware architectures for emulation of C2Q and QHT are elaborated in the next sections.

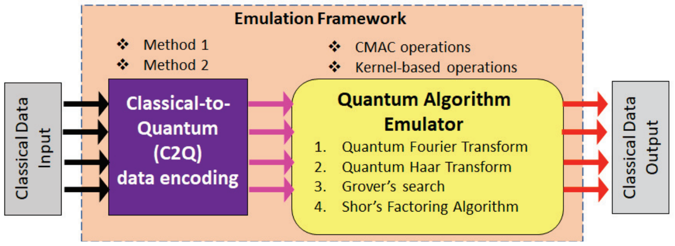

Emulation Framework
Method 1
Method 2
CMAC operations
Kernel-based operations
Classical Data Input
Classical-to-Quantum (C2Q) data encoding
Quantum Algorithm Emulator
1. Quantum Fourier Transform
2. Quantum Haar Transform
3. Grover's search
4. Shor's Factoring Algorithm
Classical Data Output

Fig. 5: Hardware architecture overview for complete emulation of quantum algorithms.

4.1.1 Hardware Architectures for C2Q Emulation

We present a hardware model, shown in Fig. 6, for emulation of the C2Q data encoding circuits in our proposed methods. To synthesize the desired state of a single qubit $|\psi\rangle$ from the zero state $|0\rangle$, the Pauli decomposition discussed in (11) can be used to define the complex coefficient pair (α, β), as in (1), in terms of the 4-tuple of parameters (r, t, θ, ϕ).

$$\alpha = r \cdot e^{i\frac{t-\phi}{2}} \cdot \cos\left(\frac{\theta}{2}\right), \quad \beta = r \cdot e^{i\frac{t+\phi}{2}} \cdot \sin\left(\frac{\theta}{2}\right) \tag{23}$$

Accordingly, for synthesizing the overall quantum state vector of N states, the C2Q hardware kernel shown in Fig. 6, iteratively in a pipelined fashion, builds the circuit structure shown in Figs. 2 and A2a (in the Appendix) for Method 1, and the circuit structure shown in Figs. 3b, 3d, and A3a (in the Appendix) for Method 2. This process continues to synthesize $\frac{N}{2}$ pairs of complex coefficients of intermediate state vectors from the set of intermediate input parameters $r_j, t_j, \theta_j, \phi_j$ where $j = 0, 1, 2, \cdots, \frac{N}{2} - 1$, see (23) and Fig. 6.

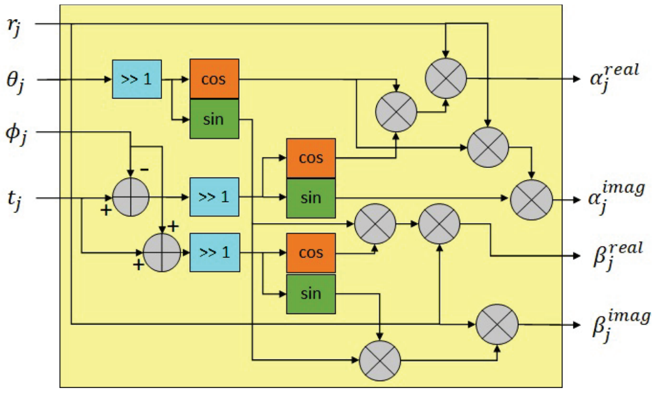
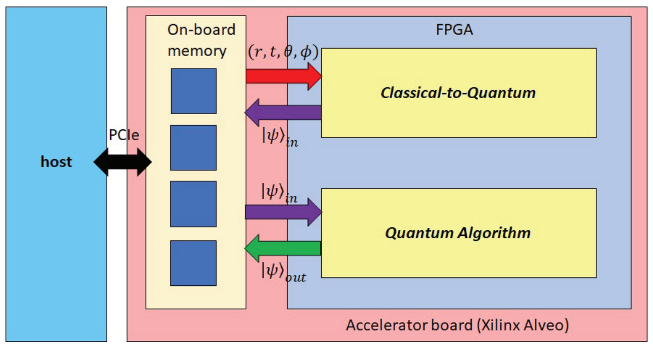


Fig. 6: Hardware architecture for C2Q kernel.

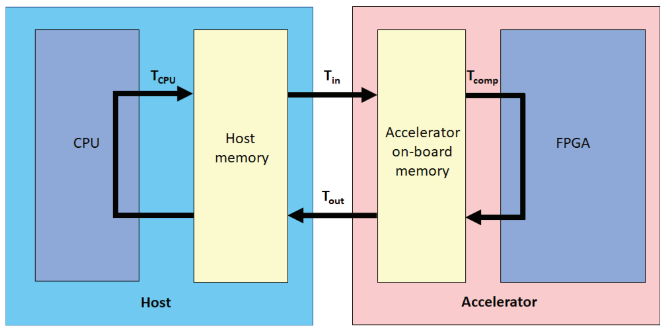
4.1.2 Hardware Architectures for QHT Emulation

As discussed previously, the QHT operations consist of input permutations, Haar transform operations, and output permutations which are denoted as P^{3D}_{in}, $U^{3-D-QHT}$, and P^{3D}_{out}, respectively for 3D-QHT. We propose hardware architectures for emulation of 3D-QHT and the corresponding architectures for P^{3D}_{in} and $U^{3-D-QHT}$ are illustrated in Figs. A6 and A7 (in the Appendix), respectively. The input permutations P^{3D}_{in} are modeled using a hardware scheduler, see Fig. A6 (in the Appendix). The scheduler works by reading the input state vector into memory, generating new indexes for data points, and then writing back to memory using generated indexes and forming the output state vector. An example of how the indices are permuted is demonstrated in Fig. A7 (in the Appendix), where an input quantum state $|x\rangle$, representing a 3D image of size $(4 \times 4 \times 4)$ pixels undergoes the 3D input permutation P^{3D}_{in} operation.

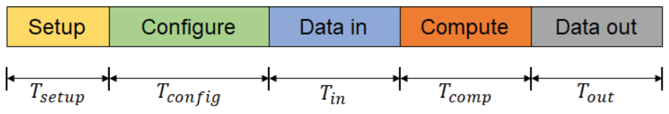
From the resulting state, the 3D Haar transformation is applied to groups of eight pixels at a time. The corresponding hardware architecture is shown in Fig. A8 (in the Appendix). Following the 3D Haar transformation, an output permutation P^{3D}_{out} is applied to the data. The operation and hardware architecture of P^{3D}_{out} is similar to P^{3D}_{in}.



(a) Emulation kernels on FPGA.



(b) Measured execution times on the host and accelerator.



(c) Timing profile for the Accelerator.

Fig. 7: Emulation execution model and timing profile for the experimental work on Xilinx Alveo FPGA.

4.2 Experimental Setup

In this section, we provide the details of each experimental implementation, i.e., FPGA, CPU, and IBM-Q Qiskit cloud-based simulator. Additionally, we present the methodology for which reported data was gathered.

4.2.1 Hardware (FPGA) Implementations

The evaluation platform used for the experimental work was an HPRC system comprising of a Xilinx Alveo U250 Data Center Accelerator connected to a host machine, see Fig. A9 (in the Appendix). The host machine has the following configuration: a 16-core, 3GHz AMD CPU, 251GB of system memory, and PCIe Gen 3 for host-to-board configuration and data communications. The Alveo U250 board contains an XCU250 FPGA that uses Xilinx stacked silicon interconnect (SSI) technology. SSI technology allows for increased density by combining 4 super logic regions (SLRs). The deployment shell that handles device bring-up and configuration over PCIe is contained within a static region of the FPGA. The remaining dynamic region is available for developers to implement custom accelerators and kernels. The dynamic region's resources consist of 1,341K look-up tables (LUTs), 2,749K registers, 2,000×36KB block RAMs, and 11,508 DSP slices. In addition, the on-board memory resources consist of four 16GB 288-pin DDR4 DIMM sockets populated with single rank DIMMs, see Fig. A9 (in the Appendix), with data transfer rates up to 2,400 MegaTransfers per second. The Vitis Unified Software from Xilinx [44] was used for design and hardware deployment. The OpenCL framework [45] was used for development of the kernels and host program. MATLAB R2020a was used for data pre-processing, post-processing, and visualizations.

The emulation execution model on Xilinx Alveo FPGA is shown in Fig. 7. The FPGA configuration and partitioning among the implemented emulation kernels is shown in Fig. 7a. The measured execution times on the system for the host and the accelerator is shown in Fig. 7b. The timing profile for the hardware accelerator is shown in Fig. 7c. The time taken by the host to perform memory allocation, setup kernel objects, kernel queues, etc. is termed as T_{setup}. The time taken to program the FPGA via PCIe is termed as T_{config}. The time taken to transfer data from the host memory to on-board memory of the FPGA is termed as T_{in}, and the time taken to transfer data from the on-board memory to the host memory is termed as T_{out}, see Figs. 7b and 7c. The compute time spent in the kernel on the FPGA is termed as T_{comp}, and it also includes the data transfer times between the FPGA and the on-board memory, see Figs. 7b and 7c.

The hardware architectures for C2Q and QHT were implemented as reconfigurable hardware kernels, *kernel_c2q* and *kernel_qht* on the FPGA. The extraction of the 4-tuple (r, t, θ, ϕ) of parameters from input dataset is performed on the host machine. The parameters and input/output state vectors $|\psi_{in}\rangle$, $|\psi_{out}\rangle$ are stored on the on-board memory and transferred to the kernel reconfigurable regions during computation. The host machine controls memory transfers and kernel execution commands via a high-speed PCIe bus. The *kernel_c2q* is executed first, which operates on the input parameters and synthesizes the input quantum state $|\psi_{in}\rangle$, which is stored on the on-board memory. The input quantum state vector is then transferred to the *kernel_qht*, which executes the parallel l-level d-dimensional QHT algorithm and produces output state vector $|\psi_{out}\rangle$, that is transferred back to on-board memory.

The kernel architectures were fully pipelined and computation operations were implemented with 32-bit floating-point arithmetic. Multi-spectral images with sizes ranging from $(16\times16\times4)$ pixels to $(32,768\times32,768\times4)$ pixels were used as input data. For these images, C2Q and 3D-QHT circuits requiring 10 to 32 qubits were emulated using the implemented emulation architectures. Equivalently sized complex data generated using a seeded random number generator was also used. Run-time results from the conducted experiments are shown in Fig. 8a and Table A1a (in the Appendix) for the C2Q kernel using positive real data (multi-spectral images) while Fig. 8b and Table A1b (in the Appendix) shows the results for the C2Q kernel using complex data. Fig. 8c and Table A1c (in the Appendix) shows the results for the 3D-QHT kernel using only positive real data (multi-spectral images), since 3D-QHT is only relevant to multi-spectral images. For both C2Q and 3D-QHT kernels, measurements of T_{in}, T_{comp}, and T_{out} were taken from host-controlled executions on the FPGA. Data packing techniques were employed to fully utilize the host-to-FPGA bandwidth and achieve optimal data transfer and compute times. The setup time T_{setup} and FPGA configuration time T_{config}, see Fig. 7c, were not included in the analysis to be consistent with CPU-based experiments. The total FPGA run-time reported is the sum of the time taken to transfer data from the host to the Alveo board, the time taken for emulation computations on the FPGA, and the time taken to transfer data back to the host, i.e., $T_{FPGA} = T_{in} + T_{comp} + T_{out}$.

In Table 2, the post-place-and-route FPGA resource utilization is reported. The FPGA area consists of a static region containing the OpenCL shell, which is responsible for interfacing the FPGA with the host and controlling host-to-accelerator memory transfers. The FPGA area also consists of a reconfigurable region comprising of the C2Q and 3D-QHT kernel spaces. The FPGA resources that are used are look-up tables (LUTs), registers (REG), Block-RAMs (BRAMs) and Digital Signal Processing (DSP) blocks. The highest total resource utilization incurred after implementation of the C2Q and 3D-QHT kernels was that for LUTs (9.48%) and BRAMs (9.29%), see Table 2. Therefore, more emulation engines, up to $\times10$, of the hardware kernels can be instantiated on a single FPGA to achieve higher throughput and faster emulation times.

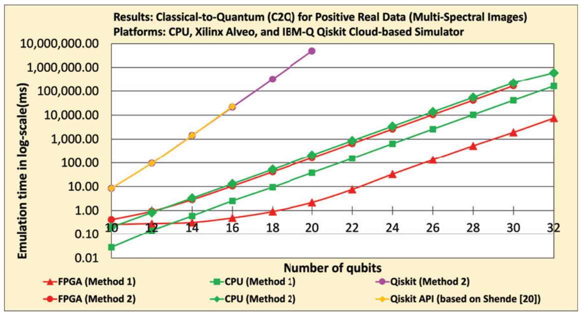
TABLE 2: FPGA resource utilization for emulation of C2Q and 3D-QHT on Xilinx Alveo.

Resource	FPGA Resource Utilization						
	Static Overlay		C2Q Kernel		3D-QHT Kernel	Total	
	C2Q (Method 1)	C2Q (Method 2)	C2Q (Method 1)	C2Q (Method 2)		C2Q (Method 1)	C2Q (Method 2)
LUT	110,382 (6.39%)	138,410 (8.02%)	4,688 (0.29%)	11,931 (0.75%)	11,478 (0.71%)	126,548 (7.39%)	161,819 (9.48%)
LUTAsMem	15,639 (1.98%)	21,035 (2.78%)	746 (0.1%)	1,174 (0.15%)	851 (0.11%)	17,236 (2.19%)	23,063 (3.04%)
REG	175,665 (5.1%)	250,988 (7.26%)	5321 (0.16%)	12,053 (0.38%)	13,546 (0.40%)	194,532 (5.66%)	276,587 (8.04%)
BRAM	203 (7.6%)	228 (8.48%)	3 (0.12%)	18 (0.73%)	2 (0.08%)	208 (7.8%)	248 (9.29%)
DSP	4 (0.03%)	7 (0.06%)	24 (0.2%)	40 (0.33%)	27 (0.22%)	55 (0.45%)	74 (0.61%)

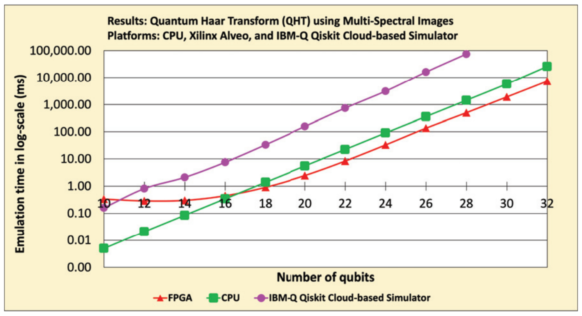
4.2.2 Software (CPU) Implementations

The software implementations were run on a high-performance computing (HPC) cluster at the University of Kansas (KU). Each of the cluster nodes was comprised of two 12-core Intel Xeon E5-2680 v3 CPUs operating at a base clock of 2.50GHz, and PCIe Gen 3.0 connectivity. Each node also featured 503GB of memory configured as 8×64GB physical DDR4 DIMMs operating at 2,133MHz.

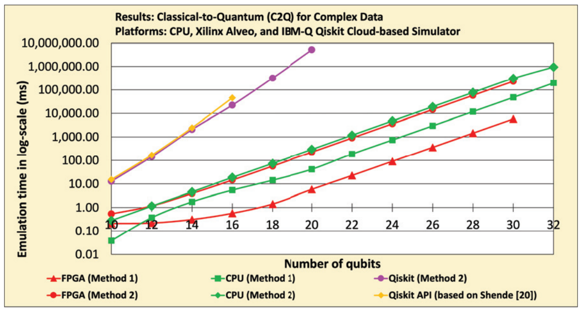
For the purposes of comparison and verification, a software-based emulator was also created for the proposed architectures using C++. Similar to the hardware (FPGA) experiments, we used the same positive real (multi-spectral images) and complex data for the CPU emulation experiments. For these data, C2Q and 3D-QHT operations required 10 to 32 qubits to be emulated on the CPU software emulator. The 4-tuple (r, t, θ, ϕ) of input parameters as well as the $|\psi_{in}\rangle$ and $|\psi_{out}\rangle$ state vectors were stored in heap-allocated memory after reading input text files and performing computations, respectively. Measurements of CPU run-times shown in Fig. 8 and Table A1 (in the Appendix) were taken from kernel executions on a single-core of the CPU on the host machine. The total CPU run-time, denoted as T_{CPU}, is the total time taken by the host (including host



(a) C2Q emulation run-times using positive real data (multi-spectral images).



(b) C2Q emulation run-times using complex data.



(c) 3D-QHT emulation run-times using positive real data (multi-spectral images).

Fig. 8: Emulation run-times for C2Q and 3D-QHT.

memory transfers) to execute the operations, see Fig. 7b. The time taken to read the input text files is not included in the reported timings.

4.2.3 IBM Quantum (IBM-Q) Qiskit Implementations

Using the same system used for the software implementations, a noise-free cloud-based simulator using Qiskit SDK (v0.38) from IBM Quantum (IBM-Q) [29] was used for implementing the proposed C2Q (Method 2) and 3D-QHT quantum circuits. The presented experimental results in Fig. 8 and Table A1 (in the Appendix) are the median of multiple runs (minimum of 10 runs) at different times of the day to amortize and mitigate the time-variant dependency of our measurements on queued simulations using IBM-Q cloud-based simulators.

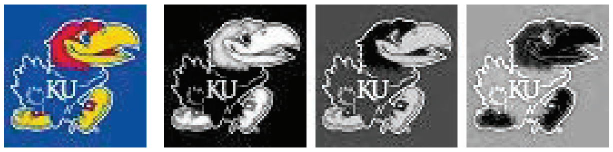
For the proposed C2Q Method 2 experiments, the circuit in Fig. 3d was implemented for the same positive real (multi-spectral images) and complex data that were used for both FPGA and CPU emulations. We compared our implementations against Qiskit API, `Initialize()`, specifically the `qiskit.circuit.QuantumCircuit.initialize()` method [30], which is based on the work by Shende *et al.* [20]. All C2Q circuits on Qiskit were given the same input data and were transpiled to 2-qubit CNOT gates and single-qubit rotation gates. From the ground state, each circuit was initialized to the input state before applying the 3D-QHT operation. The execution times for the C2Q state synthesis circuits and 3D-QHT circuits are shown in Fig. 8 and Table A1 (in the Appendix). The reported results were obtained using a circuit execution of 1 shot through the `job.result()` Qiskit API, which excludes the time taken to construct, transpile, and assemble the circuits. It is worth mentioning that Qiskit transpilation using the `Initialize()` API exceeded system memory limits for C2Q circuits requiring larger than 16 qubits while the limit was 28 qubits for 3D-QHT circuits, see Fig. 8 and Table A1 (in the Appendix).

4.2.4 Data Visualization

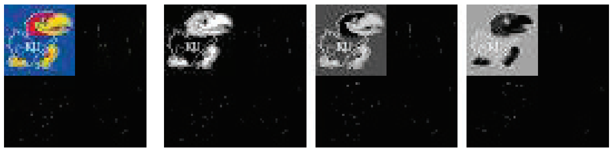
Fig. 9a shows a sample input $(64 \times 64 \times 3)$-pixel RGB-image used for the experiments. The input images were padded with zeroes to make the spectral dimension a power of 2. The image data was converted to a one-dimensional vector and normalized in MATLAB. Parameters required for C2Q were extracted from the image vector and provided as input to the FPGA, CPU, and IBM-Q simulators, where C2Q and QHT operations were performed. Post-processing, such as de-normalization, removal of padded zero data, and image reconstruction was performed in MATLAB. Fig. 9b shows the output image reconstructed after undergoing 1-level 3D-QHT. After the decomposition, the image size in each dimension is reduced by a factor of $\frac{1}{2^l}$, where l is the number of decomposition levels.

4.3 Performance Benchmarks

In this section, we present the gathered results and discuss performance comparisons among the proposed hardware-accelerated (FPGA) emulation framework in reference to software (CPU) and IBM-Q Qiskit implementations.



(a) Input RGB-image and its 3 spectral-bands.



(b) Decomposed RGB-image and its 3 spectral-bands after 1-level 3D-QHT.

Fig. 9: Dimension-reduction of $(64\times64\times3)$-pixel RGB-image using 1-level parallel (1-stage) 2D-QHT.

4.3.1 FPGA and CPU Performance Comparison

The FPGA and CPU run-times are presented in Figs. 8a and 8b for C2Q kernels using positive real (multi-spectral images) and complex data, respectively, along with Fig. 8c for the 3D-QHT kernel. The CPU outperforms the FPGA up to 12-qubit emulation and 16-qubit emulation for the C2Q and 3D-QHT kernels, respectively, as the CPU and host memory subsystem are able to take advantage of data caching. However, for larger data sizes and larger circuit emulations, the data caching is throttled, and the FPGA performance improves as it is able to take advantage of the FPGA's high bandwidth and fine-grain parallelism. To compare the performances of the FPGA and CPU, we calculated the speedup of the total FPGA execution time relative to the total CPU execution time. We observed up to $\times 21.66$ and $\times 3.49$ improvement in favor of FPGA using positive real (multi-spectral images) for C2Q (Method 1) and QHT kernels, respectively, see Fig. 8 and Table A1 (in the Appendix). The FPGA also outperforms the CPU by a factor of up to $\times 1.34$ for C2Q (Method 2) using complex data, see Fig. 8b and Table A1b (in the appendix). These results demonstrate the suitability and efficiency of our proposed methods, particularly Method 1, for hardware-accelerated (FPGA) emulations of C2Q quantum circuits. It is worth noting that our proposed Method 2 for C2Q hardware-accelerated (FPGA) emulations could still benefit from more hardware optimizations such as deep pipelining, loop fusion, super-scaling, and dense data packing/unpacking, which we will investigate in our future work.

4.3.2 FPGA and Qiskit Performance Comparison

When comparing our FPGA implementation with IBM-Q Qiskit simulator, we observed that the Qiskit simulator was bound to 16-qubit C2Q circuits using Qiskit `Initialize()` API, to 20-qubit C2Q circuits using our proposed Method 2, and to 28-qubit QHT circuits, even when supplied with 500+ GB of system memory. However, our FPGA emulation was more scalable up to 32-qubit circuits, see Fig. 8 and Table A1 (in the Appendix). For 20-qubit C2Q circuits, our FPGA emulation achieved more than 4 orders of magnitude speedup compared to the Qiskit simulator, see Fig. 8a, Fig. 8b, and Tables A1a and A1b (in the Appendix). For 26-qubit 3D-QHT circuits, the achieved FPGA emulation speedup relative to Qiskit simulator was up to $\times 148.33$, see Fig. 8c and Table A1c (in the Appendix).

4.3.3 Comparison of C2Q Method 2 with IBM-Q Qiskit

As discussed earlier, IBM Quantum (IBM-Q) [29] Qiskit API, `Initialize()`, for arbitrary state synthesis [30] is based on the reportedly most efficient recursive technique proposed by Shende *et al.* [20]. When we experimentally compared our proposed Method 2 with IBM-Q Qiskit simulator for arbitrary state synthesis, we observed identical performance in terms of circuit depth and total gate count using positive real (multi-spectral images) where both techniques synthesized identical circuits. However, our proposed Method 2 demonstrated a performance and circuit depth advantage up to 25% improvement in circuit depth when synthesizing circuits for complex data, see Fig. 10. The experimental measurements of circuit depth perfectly match our theoretical expectations, see Table 1 and Table A2 (in the Appendix).

Note that the default behavior of Qiskit API, `Initialize()`, for simulation is to directly initialize the quantum state vector with the desired values rather than to construct a C2Q circuit. In our measurements of the C2Q circuit depth, we used Qiskit transpilers for both our proposed Method 2 and Qiskit API, `Initialize()`, to actually synthesize the C2Q circuits. However, Qiskit API ran into transpilation memory limitations far sooner (16-qubit circuits) than our proposed Method 2 (20-qubit circuits), see Fig. 8a, Fig. 8b, and Tables A1a and A1b (in the Appendix). Moreover, Qiskit API, `Initialize()`, recursively constructs the C2Q circuits [20] [30], which appears to incur a far greater memory overhead compared to our non-recursive proposed Method 2, see (17) to (20). This demonstrates another advantage of our proposed Method 2 for C2Q circuit synthesis.

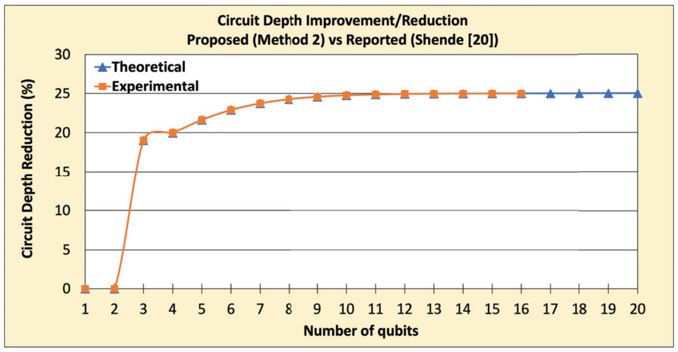


Fig. 10: C2Q circuit depth improvement/reduction.

4.4 Quantitative Comparison with existing FPGA-based emulators

A quantitative comparison of the proposed work with related FPGA-based emulators is presented in Table 3. Among the reported work [36]–[42], only few quantum algorithms have been investigated, and the corresponding quantum circuits that have been emulated are small relative to our proposed work. The low number of qubits emulated in previous works is due to the fact that the emulation techniques used were inefficiently resource-intensive and the emulator designs were bound by on-chip resources. In addition, previous emulators did not make efficient use of on-board memory, and no on-board memory utilizations were reported in their experimental works. Classical emulation of quantum algorithms is inherently memory-bound and in our work we are taking full advantage of both available on-chip and on-board memory resources, which enabled us to achieve larger-scale quantum circuit emulation compared to reported work. Information about precision, frequency, and emulation time are not reported in some of the work. The computation precision and operating frequencies are also lower compared to that in our proposed work and results. In this work, we have presented complete emulation of the QHT algorithm that includes data encoding C2Q circuits, emulating up to 32-qubit C2Q + QHT circuits with 32-bit floating point precision and operating frequency in the range of 411MHz - 414MHz.

TABLE 3: Quantitative comparison of FPGA-based quantum computing emulators.

Reported Work	Algorithm	Number of qubits	Precision	FPGA device / SOC	On-board memory (bytes)	Operating frequency (MHz)	Emulation time (sec)
Fujishima (2003) [36]	Shor's factoring	N/A	N/A	Altera APEX20K1500E-1X	N/A	80	10
Khalid et al. (2004) [37]	QFT	3	16-bit fixed pt.	Altera Stratix EP1S80B956C6	N/A	82.1	6.10E-08
	Grover's search	3	16-bit fixed pt.				8.40E-08
Aminian et al. (2008) [38]	Grover's search	3	16-bit fixed pt.	Altera Stratix EP1S80B956C6	N/A	131.3	4.60E-08
Lee et al. (2016) [39]	QFT	5	24-bit fixed pt.	Altera Stratix IV EP4SGX530KF4	N/A	90	2.19E-07
	Grover's search	7	24-bit fixed pt.			85	9.68E-08
Silva and Zabaleta (2017) [40]	QFT	4	32-bit floating pt.	AMD Xilinx ZYNQ-7000	N/A	N/A	4.00E-06
Pilch and Dlugopolski (2018) [41]	Deutsch	2	N/A	Altera Cyclone V	N/A	N/A	N/A
Suzuki et al. (2022) [42]	Image classification	6	16-bit fixed pt.	AMD Xilinx XCVU9P	N/A	250	1E-6 - 1E-2
Proposed work	C2Q	32	32-bit floating pt.	AMD Xilinx XCU250	16G	414	7.507
	QHT	32			16G	411	7.382

N/A ≡ Not Available
QFT ≡ Quantum Fourier Transform
QHT ≡ Quantum Haar Transform
C2Q ≡ Classical-to-Quantum

5 Conclusion and Future Work

Efficient emulation of quantum algorithms is necessary to investigate applications for quantum computing. In this paper, we presented an FPGA-based emulation framework for the complete emulation of quantum algorithms. The proposed emulation framework consists of hardware kernels for classical-to-quantum (C2Q) data encoding, as well as emulating the algorithm operation. The emulation framework allowed us to investigate algorithms such as quantum Haar transform (QHT) and propose optimizations. We also presented optimized quantum circuits for C2Q data encoding. We performed combined emulation of C2Q and QHT algorithms on a high-performance reconfigurable computer. Real image data was used for C2Q data encoding in the experiments and results showed accurate and correct decomposition of data after multi-dimensional QHT operation. Performance benchmarks with a software emulator and state-of-the-art quantum circuit simulator showed that the proposed hardware-based emulator is faster and more scalable. Future work will focus on improving the emulation framework with optimizations that target reducing memory I/O bottlenecks and improving inter-kernel buffering. We will also investigate quantum-to-classical (Q2C) data read-out techniques for more realistic full quantum algorithm emulation.

References

[1] Pawel Wocjan and Shengyu Zhang. Several natural bqp-complete problems. *arXiv preprint quant-ph/0606179*, 2006.

[2] Scott Aaronson. Bqp and the polynomial hierarchy. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 141–150, 2010.

[3] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[4] Google AI Quantum et al. Hartree-fock on a superconducting qubit quantum computer. *Science*, 369(6507):1084–1089, 2020.

[5] Nikitas Stamatopoulos, Daniel J Egger, Yue Sun, Christa Zoufal, Raban Iten, Ning Shen, and Stefan Woerner. Option pricing using quantum computers. *Quantum*, 4:291, 2020.

[6] IonQ. Scaling ionq's quantum computers: The roadmap.

[7] Carmen G Almudever, Lingling Lao, Robert Wille, and Gian G Guerreschi. Realizing quantum algorithms on real quantum computing devices. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 864–872. IEEE, 2020.

[8] Microsoft Quantum Team. Developing a topological qubit.

[9] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[10] Maximilian Schlosshauer. Quantum decoherence. *Physics Reports*, 831:1–57, 2019.

[11] Chi Zhang, Yanhao Chen, Yuwei Jin, Wonsun Ahn, Youtao Zhang, and Eddy Z Zhang. A depth-aware swap insertion scheme for the qubit mapping problem. *arXiv preprint arXiv:2002.07289*, 2020.

[12] Mirko Amico, Zain H Saleem, and Muir Kumph. Experimental study of shor's factoring algorithm using the ibm q experience. *Physical Review A*, 100(1):012305, 2019.

[13] David P DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik: Progress of Physics*, 48(9-11):771–783, 2000.

[14] Adam Kelly. Simulating quantum computers using opencl. *arXiv preprint arXiv:1805.00988*, 2018.

[15] M. Gokhale and L. Shannon. Fpga computing. *IEEE Micro*, 41(04):6–7, jul 2021.

[16] Amir Fijany and Colin P Williams. Quantum wavelet transforms: Fast algorithms and complete circuits. In *NASA international conference on quantum computing and quantum communications*, pages 10–33. Springer, 1998.

[17] Hai-Sheng Li, Ping Fan, Hai-ying Xia, Shuxiang Song, and Xiangjian He. The multi-level and multi-dimensional quantum wavelet packet transforms. *Scientific reports*, 8(1):1–23, 2018.

[18] Colin P Williams. *Explorations in quantum computing*. Springer Science & Business Media, 2010.

[19] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[20] Vivek V Shende, Stephen S Bullock, and Igor L Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010, 2006.

[21] Naveed Mahmud, Andrew Macgillivray, Manu Chaudhary, and Esam El-Araby. Decoherence-optimized circuits for multi-dimensional and multi-level decomposable quantum wavelet transform. *IEEE Internet Computing (IEEE IC) Special Issue on Quantum and Post-Moore's Law Computing*, 26(01):15–25, 2022.

[22] Esam El-Araby, Tarek El-Ghazawi, Jacqueline Le Moigne, and Kris Gaj. Wavelet spectral dimension reduction of hyperspectral imagery on a reconfigurable computer. In *Proceedings. 2004 IEEE International Conference on Field-Programmable Technology (IEEE Cat. No. 04EX921)*, pages 399–402. IEEE, 2004.

[23] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. Data encoding patterns for quantum computing. In *HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 22*, 2020.

[24] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. Expanding data encoding patterns for quantum algorithms. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pages 95–101, 2021.

[25] Yi Zhang, Kai Lu, Yinghui Gao, and Mo Wang. Neqr: a novel enhanced quantum representation of digital images. *Quantum Information Processing*, 12(8):2833–2860, Aug 2013.

[26] Phuc Q. Le, Fangyan Dong, and Kaoru Hirota. A flexible representation of quantum images for polynomial preparation, image compression, and processing operations. *Quantum Information Processing*, 10(1):63–84, Feb 2011.

[27] Sonika Johri, Shantanu Debnath, Avinash Mocherla, Alexandros Singk, Anupam Prakash, Jungsang Kim, and Iordanis Kerenidis. Nearest centroid classification on a trapped ion quantum computer. *npj Quantum Information*, 7(1):1–11, 2021.

[28] Naveed Mahmud, Andrew MacGillivray, Manu Chaudhary, and Esam El-Araby. Optimizing quantum circuits for arbitrary state synthesis and initialization. In *IEEE System-on-chip Conference (SOCC)*, 2021.

[29] Ibm quantum. https://quantum-computing.ibm.com. Last Accessed: October 2022.

[30] IBM Quantum Qiskit Documentation. Summary of quantum operations. https://qiskit.org/documentation/tutorials/circuits/3_summary_of_quantum_operations.html#Arbitrary-initialization. Last Accessed: October 2022.

[31] Naveed Mahmud, Bennett Haase-Divine, Andrew MacGillivray, and Esam El-Araby. Quantum dimension reduction for pattern recognition in high-resolution spatio-spectral data. *IEEE Transactions on Computers (IEEE TC)*, 71(1):1–12, 2022.

[32] List of QC simulators | Quantiki.

[33] Tyson Jones, Anna Brown, Ian Bush, and Simon C Benjamin. Quest and high performance simulation of quantum computers. *Scientific reports*, 9(1):1–11, 2019.

[34] Stavros Efthymiou, Sergi Ramos-Calderer, Carlos Bravo-Prieto, Adrián Pérez-Salinas, Diego García-Martín, Artur Garcia-Saez, José Ignacio Latorre, and Stefano Carrazza. Qibo: a framework for quantum simulation with hardware acceleration. *Quantum Science and Technology*, 7(1):015018, 2021.

[35] Constantin Gonzalez. Cloud based qc with amazon braket. *Digitale Welt*, 5(2):14–17, 2021.

[36] Minoru Fujishima. Fpga-based high-speed emulator of quantum computing. In *Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT)(IEEE Cat. No. 03EX798)*, pages 21–26. IEEE, 2003.

[37] Ahmed Usman Khalid, Zeljko Zilic, and Katarzyna Radecka. Fpga emulation of quantum circuits. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.*, pages 310–315. IEEE, 2004.

[38] Mahdi Aminian, Mehdi Saeedi, Morteza Saheb Zamani, and Mehdi Sedighi. Fpga-based circuit model emulation of quantum algorithms. In *2008 IEEE Computer Society Annual Symposium on VLSI*, pages 399–404. IEEE, 2008.

[39] Y. H. Lee, M. Khalil-Hani, and M. N. Marsono. An fpga-based quantum computing emulation framework based on serial-parallel architecture. *International Journal of Reconfigurable Computing*, 2016:5718124, Apr 2016.

[40] Agustin Silva and Omar Gustavo Zabaleta. Fpga quantum computing emulator using high level design tools. In *2017 Eight Argentine Symposium and Conference on Embedded Systems (CASE)*, pages 1–6. IEEE, 2017.

[41] Jakub Pilch and Jacek Długopolski. An fpga-based real quantum computer emulator. *Journal of Computational Electronics*, 18(1):329–342, 2019.

[42] Teppei Suzuki, Tsubasa Miyazaki, Toshiki Inaritai, and Takahiro Otsuka. Quantum ai simulator using a hybrid cpu-fpga approach. https://arxiv.org/abs/2206.09593, 2022. Last Accessed: October 2022.

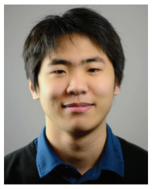
[43] Naveed Mahmud, Esam El-Araby, and David Caliga. Scaling reconfigurable emulation of quantum algorithms at high precision and high throughput. *Quantum Engineering*, 1(2):e19, 2019.

[44] Vinod Kathail. Xilinx vitis unified software platform. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, page 173–174, New York, NY, USA, 2020. Association for Computing Machinery.

[45] Aaftab Munshi. The opencl specification. In *2009 IEEE Hot Chips 21 Symposium (HCS)*, pages 1–314. IEEE, 2009.

Esam El-Araby is an associate professor in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. He received his Ph.D. and M.Sc. degrees in Computer Engineering from the George Washington University (GWU), USA, in 2010, and 2005 respectively. Dr. El-Araby worked at the High-Performance Computing Laboratory (HPCL) at GWU as well as the National Science Foundation (NSF) Center for High-Performance Reconfigurable Computing (NSF-CHREC). His research work was mainly funded by organizations such as DoD, DARPA, NSF, and NASA. Dr. El-Araby is the recipient and the Principal Investigator (PI) of several significant awards from NSF such as the prestigious award of Faculty Early Career Development Program (NSF-CAREER) and the Major Research Instrumentation (NSF-MRI) award. His primary research interests are in computer architecture, reconfigurable computing, quantum computing, quantum communications, reversible computing, heterogeneous computing, biologically-inspired and neuromorphic architectures, and evolvable hardware.

Naveed Mahmud is an Assistant Professor in the department of Computer Engineering and Sciences (COES) at Florida Institute of Technology. He received his Ph.D. in Electrical Engineering in 2022 from the University of Kansas (KU), USA with honors in the department of Electrical Engineering and Computer Science (EECS). Dr. Mahmud's primary research interests are future and emerging computing architectures such as quantum computing and reconfigurable computing. His work involves developing reconfigurable emulation architectures for quantum algorithms, optimizing quantum algorithms for efficient implementation on near-term quantum devices, and developing applications for quantum computing. His publication record includes several peer-reviewed journal articles and conference papers. Dr. Mahmud also worked as a Senior Design Verification Engineer for Ulkasemi, a top semiconductor services firm in Bangladesh with off-shore design centers in the USA. He was responsible for leading the Digital Design Verification team in addition to developing automated, self-checking testbenches for ASICs.



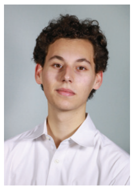
Mingyoung Joshua Jeng is a graduate student in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU). Joshua has previously received a B.Sc. degree in Engineering Physics from KU in Spring 2022. He is currently working as a Graduate Research Assistant at KU. Joshua was the recipient of KU's Feaster Scholarship in 2021. He was also a former supplemental instructor for Introduction to Digital Logic Design in the department of EECS at KU. Joshua's current research interests are quantum computing, specifically the development and implementation of quantum algorithms. Joshua's publication record includes one conference paper and one conference poster.

Andrew MacGillivray is an undergraduate student in Computer Engineering in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. Andrew has acquired industry experience through his ongoing employment as a programmer in the aerospace sector, where he focuses on the development of business-administration tools. His publication record includes five journal articles and one conference paper.

Manu Chaudhury is a Ph.D. student in the department of Electrical Engineering and Computer Science (EECS), at the University of Kansas (KU), USA. Manu received the MSEE degree in Electrical Engineering from The University of North Carolina, Charlotte, NC, USA, in 2019. He currently works as a Graduate Research Assistant and a Graduate Teaching Assistant in EECS at KU. He is teaching the labs of both Digital Systems Design and Introduction to Digital Logic Design. Manu's current research focuses on quantum computing and emulation of quantum algorithms on FPGAs. His publication record includes one journal article and three conference papers.

Md. Alvir Islam Nobel is a Ph.D. student in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. Alvir completed his B.Sc. degree in Electrical and Electronic Engineering from Ahsanullah University of Science and Technology (AUST), Bangladesh, in 2019. Alvir was the recipient of KU's EECS Engineering Scholarship in 2021. He currently works as a Graduate Research Assistant and a Graduate Teaching Assistant in EECS at KU. He is teaching the labs of both Digital Systems Design and Introduction to Digital Logic Design. Alvir's current research is focused on reconfigurable computing, and optimization of quantum algorithms.

SM Ishraq Ul Islam is currently a Masters student in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. He completed his undergraduate degree in Electrical and Electronic Engineering from the Islamic University of Technology, Bangladesh. He is currently serving as a GTA at KU and conducts labs for the Embedded Systems course. His current research interests include quantum computing and reconfigurable computing.

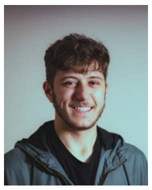


David Levy is an undergraduate student in Computer Engineering in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. David has been the recipient of the KU Achievement Scholarship for two consecutive years in 2021 and 2022, as well as the Garmin Excellence Scholarship in the 2022-2023 academic year, and also made the KU Honor Roll in both semesters of the 2021-2022 academic year. David's current research interests include quantum computing and the emulation of quantum algorithms. David's publication record includes one conference paper and one conference poster.

Dylan Kneidel is an undergraduate student in Computer Science in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. Dylan is a current member of the SELF Engineering Leadership Fellows Program in the class of 2026, and his current interests include computer architecture, quantum computing, and programming language design.

Madeline R. Watson is an undergraduate student in Computer Science and class of 2026 SELF Fellow in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. She is a recipient of the Regional Aspirations in Computing award in 2022. Madeline's research interests include circuit design and quantum computing.

Jack G. Bauer is an undergraduate student in Computer Science and class of 2026 SELF Fellow in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. He has worked on multiple projects in Java, C# and Python. Jack's research interests include quantum computing and artificial intelligence.



Andrew E. Riachi is an undergraduate student in Computer Engineering in the department of Electrical Engineering and Computer Science (EECS) at the University of Kansas (KU), USA. He is a 2019 National Merit Finalist, the recipient of an EECS Undergraduate Achievement Award, and a recipient of the Garmin Excellence Scholarship. His interests include low-level and embedded software, compilers, and computer architecture.