FPGA Emulation of Quantum Circuits

Ananya Nawale[*] and Simran Sinha[†]
Department of Physics
Indian Institute of Technology Bombay

Quantum computing is an promising technology with potential applications in various fields, from cryptography to drug discovery. However, building a practical quantum computer is a daunting task, as it requires the development of new hardware, software, and algorithms. In the meantime, the emulation of quantum circuits on classical hardware, such as Field-Programmable Gate Arrays (FPGAs), can provide a way to experiment and test quantum algorithms. This report presents the design and implementation of a FPGA-based quantum circuit emulator that can simulate arbitrary quantum circuits with up to 3-4 qubits. The emulator uses a custom VHDL implementation that allows for high-level abstraction of quantum gates and circuits, while providing efficient hardware utilization and parallelism. The FPGA architecture used in the emulator is described in detail, including the programmable logic blocks, routing resources, clocking, and I/O interfaces. The emulator supports a variety of quantum gates, including the Hadamard gate and phase gate, and can simulate the behavior of quantum circuits with multiple gates and measurements. The report also discusses the potential applications of the emulator, including the verification and testing of quantum algorithms, and the development of quantum-inspired classical algorithms.

I. INTRODUCTION

Quantum computing is an emerging field that has the potential to revolutionize the way we approach problems that are too complex for classical computers to solve efficiently. However, developing and testing quantum algorithms on actual quantum hardware can be challenging and expensive. Hence, simulating quantum circuits on classical hardware, such as FPGAs, is becoming an increasingly popular alternative.

One of the key components of quantum computing is the quantum gate, which operates on qubits and enables the manipulation of quantum information. In this project, we aim to implement several fundamental quantum gates and circuits, including the Hadamard gate, controlled NOT (CNOT) gate, Toffoli gate, swap gate, and phase gate, on an FPGA using the Verilog hardware description language/VHDL. These gates are essential building blocks for constructing more complex quantum circuits, and by implementing them on an FPGA, we can simulate the behavior of quantum circuits efficiently and cost-effectively.

Our ultimate goal is to implement the Quantum Fourier Transform (QFT) on the FPGA, which is a crucial component of many quantum algorithms, including Shor's algorithm for factoring large numbers. By implementing the QFT on an FPGA, we can simulate the behavior of a quantum circuit up to a certain number of qubits and demonstrate the potential of FPGAs as a tool for simulating quantum algorithms.

II. FPGA ARCHITECTURE OR VHDL BASED SIMULATION OF QUANTUM CIRCUITS

The field of quantum computing is still relatively new, and hardware modeling of quantum processes is complex and challenging. It is likely that the model itself will require many changes and revisions as understanding of quantum computing continues to evolve. This can result in multiple re-runs of simulations, which can be time-consuming and resource-intensive. Therefore, emulators that can perform fast simulations and can be easily re-programmed to account for changes in the quantum model as well as the design itself are essential.

FPGA-based hardware emulators provide such capabilities, enabling efficient simulation of quantum circuits and algorithms. Using the quantum circuit model, which is an intuitive way of constructing quantum algorithms, greatly assists in the development of new algorithms. The emulator combines important quantum computing concepts and a solid development environment to serve as a development and test-bed for quantum algorithms. This approach allows to explore the behavior of quantum algorithms in a cost-effective and efficient manner, accelerating the pace of progress in the field of quantum computing [1].

The process of emulating quantum circuits using classical hardware description languages poses a significant challenge due to the inherent differences between quantum and classical circuits. Although the quantum circuit model can be used to create an analog for quantum computing in the classical domain, it still has substantial differences from classical circuits. Therefore, the aim is to develop an efficient method for representing quantum circuits using a classical hardware description language that can be synthesized on an FPGA. This process involves mapping quantum computation concepts to the digital domain in a way that effectively captures the unique features of quantum circuits. By addressing this problem,

[*] 21d170004@iitb.ac.in
[†] 210260051@iitb.ac.in

we can create effective hardware emulators for quantum circuits that can be easily re-programmed to account for small changes in the model of quantum processes or the design itself.

The second major challenge in emulating quantum circuits using FPGAs is to ensure that the available resources on the FPGA are used as efficiently as possible. In general, emulating quantum circuits requires a large number of computational resources, especially for highly entangled systems. This is because entangled systems require exponential amounts of computation to perform quantum evolution. However, FPGAs have an advantage in this regard, as they contain a large number of logic cells, which can be used to emulate large quantum circuits with entangled states efficiently in terms of computation time. In addition, multiple FPGAs can be combined to emulate even larger quantum circuits. By using FPGAs, it is possible to simulate complex quantum circuits with high accuracy while minimizing the computational resources required.

Simulation of quantum circuits using a hardware description language such as VHDL is being proposed. This technique uses analogies between the quantum circuit model and classical circuits to construct and simulate the quantum circuits. As the quantum circuit model breaks down the quantum transform into quantum gates and quantum bits, the VHDL approach can incorporate these architectures directly as they are congruent analogues to classical gates and bits respectively.

Qubits are described as two complex numbers (using the real keyword in VHDL). The gates are constructed with qubits as inputs and outputs and the transform of the gate is described using behavioral VHDL. The quantum circuit can be constructed now by combining the gates using structural VHDL [2].

III. THEORETICAL BACKGROUND

Quantum computing is based on the properties of quantum mechanics, namely, superposition and entanglement. Superposition allows a quantum state to be in more than one basis state simultaneously, whereas entanglement is the strong correlation between multiqubit (quantum bit) basis states in a quantum system. Superposition and entanglement facilitate massive parallelism which enables exponential speed-ups to be achieved in the well-known integer factoring and discrete logarithms algorithms and quadratic speed-ups in solving classically intractable brute-force searching and optimization problems [3].

Quantum algorithms are built using the quantum circuit model, which is based on quantum mechanics principles. This model involves the use of qubits, which are quantum bits, and the operations that can be performed on them. In general, quantum algorithms follow a specific process flow structure. The computation process begins with the preparation of a quantum state, which is set to a specific initial state. Unitary transformations, represented by quantum gates are applied to it according to the required operations of the algorithm. The unitary transformations on the quantum state allow the algorithm to perform complex calculations and solve problems that are difficult or impossible for classical computers. This is because qubits can exist in multiple states at once, which allows for parallel computation. Finally, the quantum state is measured, which results in the collapse of the qubits into classical bits. The measurement provides the output of the algorithm, which can be used to solve a problem or perform a specific task.

A. Quantum Information Representation

In classical computing, digital circuits operate using binary values of 0 and 1. However, in quantum computing, the basic unit of information is called a qubit and it can exist in a superposition of states. A qubit can be represented by a vector of complex numbers as bearers of information, where each number corresponds to the probability amplitude of finding the qubit in a particular state. These complex numbers cannot be directly represented by classical bits. To represent a single qubit, a quantum computer would require a large number of classical bits, depending on the precision required to represent the complex numbers [4].

More formally, the states of the quantum system belong to a vector space over complex numbers in which there exists an inner product of vectors, usually referred to as the Hilbert Space H. In denoting these vectors, commonly used is the Dirac bra-ket notation.

B. Quantum Bits (Qubit)

The basic units of quantum information can be viewed as simple two-state systems, such as magnetic spin of plus/minus one half. The state of a spin is given as a continuous quantity represented by two real numbers. It is exactly this continuity in spin representation that contributes to the ability of storing the infinite classical information by a single quantum system. Quantum bits defined this way are commonly referred to as qubits.

Binary qubits have two computational base states denoted as $|0\rangle$ and $|1\rangle$. Unlike classical bits, quantum bits are in a linear superposition of the basis states $|0\rangle$ and $|1\rangle$

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \tag{1}$$

where α and β are complex coefficients related as

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2}$$

The superposition phenomena, by which the qubits simultaneously exist in states $|0\rangle$ and $|1\rangle$ is explained by considering $|\alpha|^2$ and $|\beta|^2$ as probabilities of being in $|0\rangle$ and $|1\rangle$ respectively. However, when a measurement is performed on a qubit, it collapses to either of the two basis states.

C. Tensor/Kronecker product

The tensor product, also known as the Kronecker product, is a mathematical operation used to combine two or more quantum states to create a larger system as well as to describe multi-qubit quantum transformations. A separable quantum state vector can be expressed as a tensor product of two or more single-qubit states, whereas an entangled state cannot be expressed as the tensor product of two or more single-qubit states. This is because entangled states are created through quantum entanglement, which is a phenomenon where the quantum states of two or more particles become correlated in such a way that the states of the individual particles cannot be described independently.

In quantum circuits, the tensor product is used to describe the evolution of the quantum state as it passes through the circuit. The tensor product of the individual quantum gates applied to each qubit in the circuit describes the overall transformation applied to the quantum state. The tensor operation on any arbitrary two 1-qubit transformations is:

$$\begin{pmatrix} a_0 & a_1 \\ a_2 & a_3 \end{pmatrix} \otimes \begin{pmatrix} b_0 & b_1 \\ b_2 & b_3 \end{pmatrix} = \begin{pmatrix} a_0b_0 & a_0b_1 & a_1b_0 & a_1b_1 \\ a_0b_2 & a_0b_3 & a_1b_2 & a_1b_3 \\ a_2b_0 & a_2b_1 & a_3b_0 & a_3b_1 \\ a_2b_2 & a_2b_3 & a_3b_2 & a_3b_3 \end{pmatrix} \quad (3)$$

D. Quantum circuit Model

A quantum algorithm is a set of instructions for performing a computation using quantum mechanical systems, usually qubits. These instructions describe a sequence of quantum operations or transformations that manipulate the quantum states of qubits in order to perform a specific task or solve a particular problem. The quantum circuit model is a common method used to represent quantum algorithms. In this model, quantum gates are used to manipulate the qubits, and these gates are represented by unitary matrices.

All unitary matrices are invertible and the products of unitary matrices as well as the inverse of unitary matrix are unitary. An N-by-N matrix U is unitary if $UU^\dagger = U^{dagger}U = I_N$, where $U^\dagger$ is the adjoint(conjugate transpose) of U. Since all quantum transformations are reversible, quantum gate operations can always be undone.

1. Hadamard Gate

One of the most useful single qubit quantum gates. It operates by placing the computational basis state into superposition of basis states with equal probability. The Hadamard transform can be represented by the following unitary matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (4)$$

2. Controlled-NOT Gate

Controlled NOT gate accepts two quantum bits: a control qubit $|\zeta\rangle$ and a target qubit $|\psi\rangle$ and produces the outputs: $|\zeta\rangle$ and $|(\zeta \oplus \psi)\rangle$ where $\oplus$ is the XOR operation. Transformation matrix:

$$U_{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (5)$$

That is, if $|\zeta\rangle = |0\rangle$, then $|\psi\rangle$ is inverted

3. Phase Gate

The operation of the phase shift gate on the single qubit is defined as: $|0\rangle \longrightarrow |0\rangle$ and $|1\rangle \longrightarrow e^{\iota\phi}|1\rangle$. The matrix for phase gate is:

$$U_S = \begin{pmatrix} 1 & 0 \\ 0 & e^{\iota\phi} \end{pmatrix} \quad (6)$$

E. Quantum Fourier Transform

Fourier Transform is a reversible transformation that converts signals from time/spatial domain to frequency domain and vice versa. Defined for discrete signals:

$$X(f) = \sum_{n=0}^{N-1} x_n e^{-\iota 2\pi (kn/N)} \quad (7)$$

The quantum Fourier transform is a transformation on qubits and is the quantum equivalent of the discrete Fourier transform. It operates on a quantum state and transforms it into a different quantum state. Compared to the classical Fourier transform, the QFT can perform the transformation with exponentially fewer operations. However, the execution time of the algorithm does not decrease when classical data is used. This is because the quantum computer does not allow parallel read-out of all quantum state amplitudes. Furthermore, it is difficult to instantiate the desired input state amplitudes to be Fourier-transformed.

QFT is a key component of many quantum algorithms, including Shor's algorithm for integer factorization and the discrete logarithm problem, which are used for breaking classical encryption. It is also used in Simon's algorithm for finding periodicity in a function and in Hallgren's algorithms for solving certain diophantine equations. These algorithms provide significant speed-up over classical algorithms for the same problems.

To compute Fourier transform in quantum domain, discrete signal samples are encoded as the amplitude sequences of a quantum state vector which is in superposition of basis states. An n-qubit QFT operation which transforms an arbitrary superposition of computational basis states is expressed in

$$|\psi\rangle = \frac{1}{2^n}\sum_{j=0}^{2^n-1} f(j\Delta t)\,|j\rangle \xrightarrow{QFT} \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}\sum_{j=0}^{2^n} f(j\Delta t)e^{2\pi\iota(jk/2^n)}\,|k\rangle \tag{8}$$

As the requirement for a valid quantum state, $|\psi\rangle$ must be normalized such that it fulfills $\sum_{j=0}^{2^n-1}|f(j\Delta t)|^2 = 1$. From (8), the term $j/2^n$ is a rational number in the range of $0 \leq j/2^n < 1$. As qubit representation is typically used in computations, the j in base-10 integer is redefined in base-2 notation as individual bit such that the binary fraction form as expressed in (9) can be conveniently adopted:

$$(j)_{10} \equiv (j_1 j_2 \cdots j_n)_2 = (2^{n-1}j_1 + 2^{n-2}j_2 + \cdots + 2^0 j_n)_{10}$$

$$= 2^n(2^{-1}j_1 + 2^{-2}j_2 + \cdots 2^{-n}j_n)_{10} = 2^n(0.j_1 j_2 \cdots j_n)_2 \tag{9}$$

With some algebraic manipulations, the QFT equation:

$$QFT_{2^n}\,|j\rangle = \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{2\pi\iota(jk/2^n)}\,|k\rangle =$$

$$\frac{1}{\sqrt{2^n}}(|0\rangle + e^{2\pi\iota 0\cdot j_n}|1\rangle)(|0\rangle + e^{2\pi\iota 0\cdot j_{n-1}j_n}|1\rangle)\cdots(|0\rangle + e^{2\pi\iota 0\cdot j_1 j2\cdots j_n}|1\rangle) \tag{10}$$

Since the term $e^{2\pi\iota 0\cdot j_1}$ produces either -1 if $j_1 = 1$ or +1 otherwise, Hadamard computation on the first qubit results in $(\frac{1}{\sqrt{2}})(|0\rangle + e^{2\pi\iota 0\cdot j_1}|1\rangle)$. QFT circuit consists of three types of elementary gates which are Hadamard gate, Controlled phase-shift gate, controlled Rotation gate and swap gate. [5].

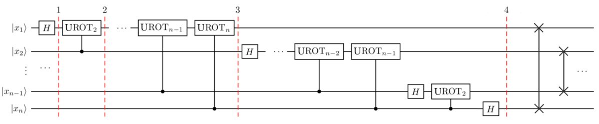

Fig: Quantum circuit model for n-qubit QFT

The size of a QFT circuit grows exponentially as the number of input qubits increases. An n-qubit QFT involves $\sum_{k=1}^{k+1}$ unitary transformations and could process up to 2^n input samples in one evaluation (provided the input samples are encoded as the amplitude sequences of a superposition of computational basis states).

IV. ARCHITECTURE OF PROPOSED FPGA EMULATION MODEL

A. Fixed Point Representation of Qubits

A quantum bit is implemented using Equation (1). Thus, we need to store the values of α and β to describe a qubit. The finite precision description of α and β introduces imprecision errors, as gates involve operations such as addition and multiplication on α and β. α and β are implemented by storing their real and imaginary coefficients into 2 registers. The schematic for storing these coefficients is described below:

1 Sign Bit	1 Pre-Decimal Bit	N-1 Post-Decimal Bits

Fig: Fixed-point representation of coefficients

This schematic is used to store every signed floating point number used in this implementation. Regarding the precision, keeping space and time constraints in mind, the floating point numbers have been stored in a 17 bit array. Nonetheless, the emulator has been designed in a modular way, meaning that the size of the fractional part can be adjusted as required. This is advantageous for experiments that deal with precision and fault-tolerance of quantum algorithms as it allows for easy testing of various precision levels without affecting the rest of the system.

Verilog possesses inherent mechanisms for executing fundamental arithmetic operations, namely addition, subtraction, and shift operations. However, the language does not provide support for operations such as regular multiplication, complex multiplications, and the like. In order to facilitate such operations, it is necessary to develop customized modules. The block diagram for complex multiplication along with variable description is provided below:

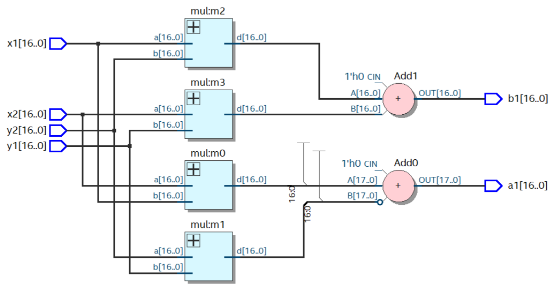

Fig: Block Diagram for complex multiplication

x1 — Real coefficient of Complex number 1
y1 — Imaginary coefficient of Complex number 1

x2 — Real coefficient of Complex number 2
y2 — Imaginary coefficient of Complex number 2
a1 — Real coefficient of Output
b1 — Imaginary coefficient of Output

B. Hadamard

The Hadamard gate when applied to a single qubit in an arbitrary quantum state generates a transformation resulting in a new qubit state. To implement the gate's functionality, it is necessary to provide four coefficients as input, apply corresponding operations to these coefficients, and finally generate a set of four coefficients as output. The Block diagram for a 1 qubit hadamard gate along with the variable description is provided below:

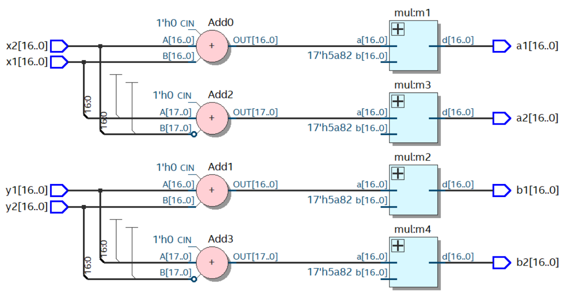

Fig: Block Diagram for Hadamard gate

x1 — Real coefficient of Input $|0\rangle$
y1 — Imaginary coefficient of Input $|0\rangle$
x2 — Real coefficient of Input $|1\rangle$
y2 — Imaginary coefficient of Input $|1\rangle$
a1 — Real coefficient of Output $|0\rangle$
b1 — Imaginary coefficient of Output$|0\rangle$
a2 — Real coefficient of$|1\rangle$
b2 — Imaginary coefficient of Output$|1\rangle$

C. Phase Gate

The general implementation of a phase gate necessitates the utilization of sine and cosine functions, which cannot be directly synthesized on an FPGA. Although they can be approximated using Taylor expansion series, such approximations remain valid only for certain angles. However, for the specific purpose of constructing the QFT module, we have developed two unique types of phase gates, R2 and R3, that are precise and enough for our requirements. They are also equipped with an enable latch. Their block diagrams are showcased below:

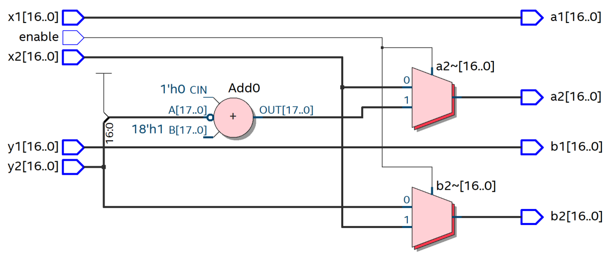

Fig: Block Diagram for R2 gate

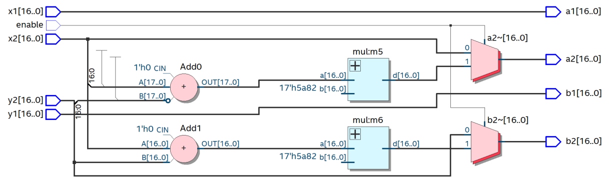

Fig: Block Diagram for R3 gate

Both of them have the same variable description:
x1 — Real coefficient of Input $|0\rangle$
y1 — Imaginary coefficient of Input $|0\rangle$
x2 — Real coefficient of Input $|1\rangle$
y2 — Imaginary coefficient of Input $|1\rangle$
a1 — Real coefficient of Output $|0\rangle$
b1 — Imaginary coefficient of Output$|0\rangle$
a2 — Real coefficient of$|1\rangle$
b2 — Imaginary coefficient of Output$|1\rangle$

D. QFT

Emulation of QFT using hadamard and phase gates as described here restricts its input quantum state to only pure computational basis state, implying that superposition is not included in the modelling. Consequently, the input for the QFT module is restricted to the state of a three-qubit system in the computational basis, which can be stored in a three-bit unsigned register represented by the variable x. Given below is the block Diagram for QFT for pure computational basis input. The block diagram for QFT with a pure computational basis input is presented below, with the omission of the output diagram. This is because the QFT output comprises the coefficients generated by the tensor product of the diagram depicted in Figure 1.

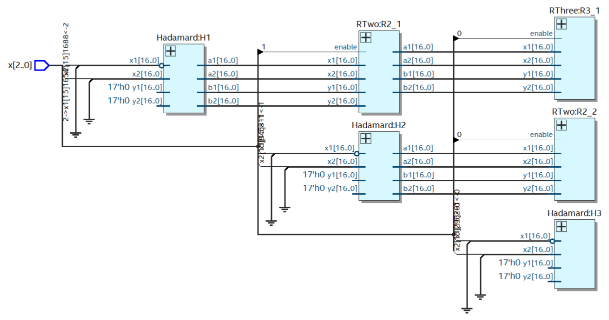

Fig 1: Block Diagram for QFT for pure computational basis input

The block diagram below illustrates the computational method employed to obtain the final outputs via tensor product.

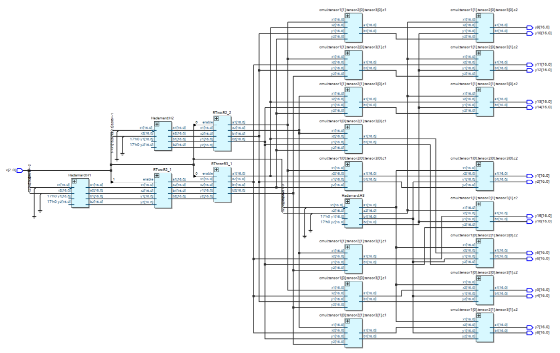

Fig2 : Block Diagram for QFT with Tensor Product for pure computational basis input

Variable Description is as follows:

input reg [2:0] x — Input state into the QFT module $|0\rangle$

y1— Real coefficient in FT basis of $|000\rangle$
y2 — Imagniary coefficient in FT basis of $|000\rangle$
y3— Real coefficient in FT basis of $|001\rangle$
y4 — Imagniary coefficient in FT basis of $|001\rangle$
y5— Real coefficient in FT basis of $|010\rangle$
y6 — Imagniary coefficient in FT basis of $|010\rangle$
y7— Real coefficient in FT basis of $|011\rangle$
y8 — Imagniary coefficient in FT basis of $|011\rangle$
y9— Real coefficient in FT basis of $|100\rangle$
y10 — Imagniary coefficient in FT basis of $|100\rangle$
y11— Real coefficient in FT basis of $|101\rangle$
y12 — Imagniary coefficient in FT basis of $|101\rangle$
y13— Real coefficient in FT basis of $|110\rangle$
y14 — Imagniary coefficient in FT basis of $|110\rangle$
y15— Real coefficient in FT basis of $|111\rangle$
y16 — Imagniary coefficient in FT basis of $|111\rangle$

Hence, our current QFT module is restricted to calculating QFT solely for states that exist exclusively in the computational basis. However, with the utilization of our complex multiplier, it is possible to develop a module capable of computing QFT for states in superposition. Unfortunately, it becomes extremely complicated to look at and determine the logic. Thus, a simple flowchart illustration of the logic has also been provided.

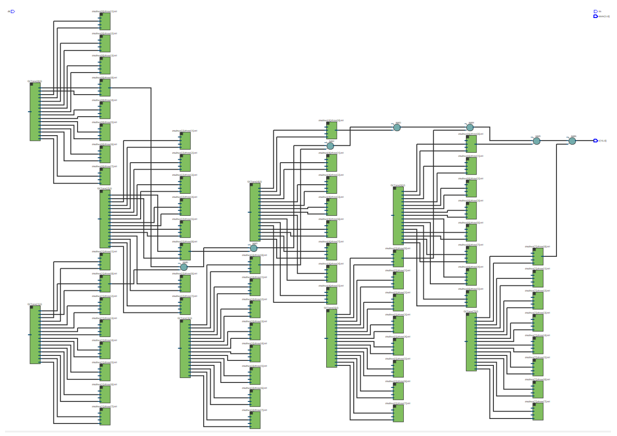

Fig : Block Diagram for complete implementation of QFT

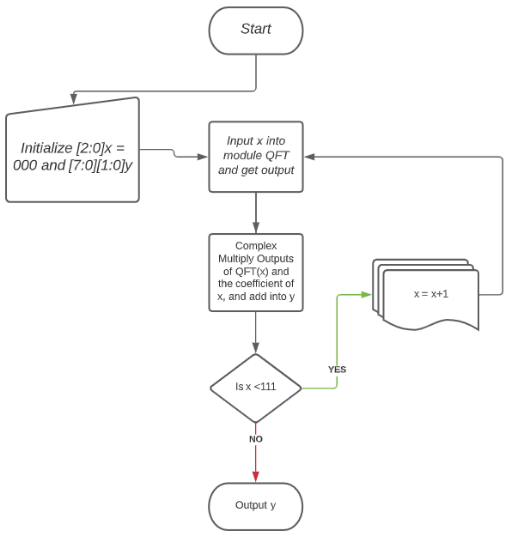
Start
Initialize [2:0]x = 000 and [7:0][1:0]y
Input x into module QFT and get output
Complex Multiply Outputs of QFT(x) and the coefficient of x, and add into y
x = x+1
Is x <111
YES
NO
Output y

Fig : Logic Flowchart for complete implementation of QFT

V. EXPERIMENTAL IMPLEMENTATION

A. Demonstration of QFT coefficients in Binary

To showcase the coefficients of states of QFT with 16-bit precision in binary format(5-digit precision in decimal), we utilized a breadboard and LEDs connected to the FPGA. Binary values of the coefficients were displayed using LEDs, with the red LED representing the sign bit, and the green LED representing the pre-decimal portion. The remaining LEDs represented the decimal portion of the coefficient. By representing the quantum state amplitudes in binary format, we were able to display them using the LEDs. The binary values were then converted to decimal values to demonstrate the precise coefficients of states. This demonstration allowed for a visual representation of the complex amplitudes involved in the QFT calculation and provided an opportunity to observe the quantum phenomenon of superposition. It served as a practical application of the QFT emulation and showcased the precision and accuracy of the emulation framework. By showcasing the coefficient states of QFT in binary format, we were able to highlight the power of quantum computing in processing large amounts of data more efficiently than classical computing.

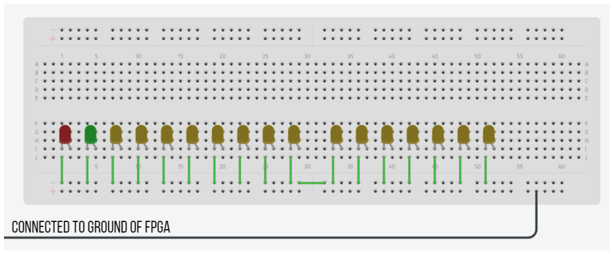

Fig : Experimetal Setup for showcasing coefficients.

B. Demonstration of Superposed States of QFT

To visualize the output of the QFT implementation, we used a square wave signal, where the width of each pulse is proportional to the probability of that state appearing in the output. We kept the gap between each pulse constant for ease of observation. However, to obtain the exact value of each coefficient, we referred to the coefficient display using LEDs. By observing the pulse widths, we could infer the probabilities of each state and hence, verify the correctness of the QFT implementation. This method of visualization proved to be a useful tool in understanding the behavior of the QFT algorithm and the superposed states it generates.

VI. RESULTS

Displayed below are the ModelSim simulation's of the QFT module with inputs restricted to the pure computational basis. As each of the inputs has an impact on all of the outputs, the cause effect arrows have been omitted to circumvent clutter and facilitate the simulation's comprehension.

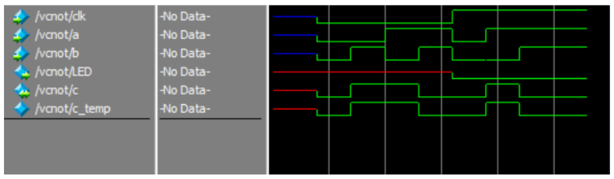

Fig : ModelSim digital simulation of CNOT gate

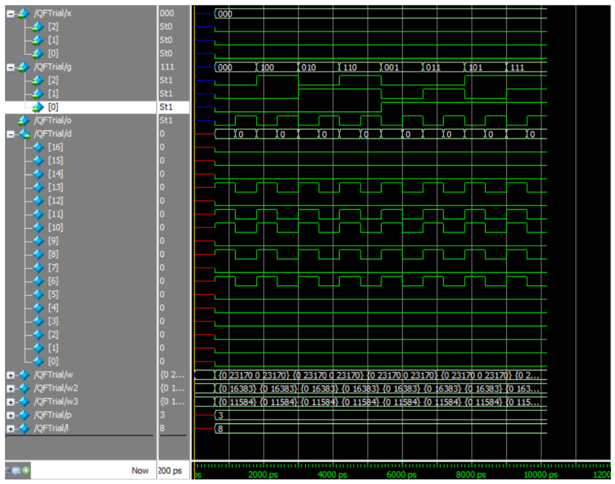

Fig : ModelSim digital simulation of QFT for input state $|000\rangle$

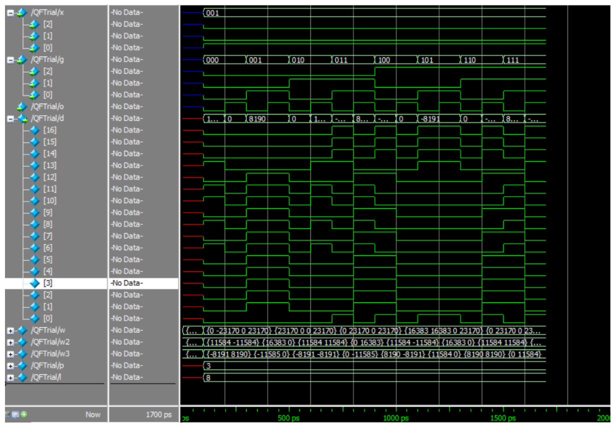

Fig : ModelSim digital simulation of QFT for input state $|001\rangle$

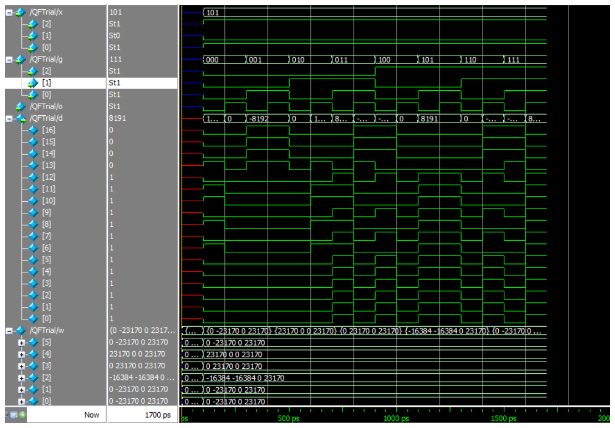

Fig : ModelSim digital simulation of QFT for input state $|101\rangle$

Link to all the codes for the project: https://github.com/infinite55/EE224$_{project.git}$

VII. CONCLUSIONS

Efficient resource utilization is critical for FPGA-based implementations especially for emulating quantum computing applications as they typically exhibit exponential resource requirement with increasing number of qubits. To address this, we propose a baseline FPGA emulation framework that focuses on optimizing the datapath design based on the conventional state vector model as well as an effective methodology that facilitates accurate modelling of quantum algorithms for FPGA emulation. The proposed architecture is a serial-parallel design that efficiently utilizes resources on the FPGA. It leverages parallel processing to perform operations on multiple qubits simultaneously while minimizing the number of required operations. We also describe techniques for reducing the number of gates and optimizing the mapping of quantum circuits to the FPGA.

However, experimental results obtained in this work show that it is difficult to realize a scalable and flexible emulation platform for large qubit size real-world quantum system using the approach that applies existing state vector quantum models. This is due to the exponentially large memory requirements for storing the entire state vector. Hence, it suggests that a model with a more effective data structure to represent quantum systems is required to overcome this challenge. With a more efficient modelling technique, FPGA can represent a more efficient emulation strategy of quantum systems.

ACKNOWLEDGMENTS

We extend our heartfelt appreciation to our course instructor, Prof. Pradeep Sarin, for providing us with the opportunity to work on this project and guiding us throughout the process. We also express our gratitude to Prof. Himadri Dhar, for providing us with valuable insights and feedback, which greatly improved the quality of our work. Their guidance and support helped us to better understand the theoretical concepts of quantum circuits and emulation. We would also like to thank the teaching assistants of the course for their assistance in various aspects of the project. Their prompt response and willingness to help made the project progress smoother.

[1] A. U. Khalid, Z. Zilic, and K. Radecka, in *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* (IEEE, 2004) pp. 310–315.

[2] K. Skahill, *VHDL for programmable logic* (Addison-Wesley Longman Publishing Co., Inc., 1996).

[3] Y. H. Lee, M. Khalil-Hani, and M. N. Marsono, International Journal of Reconfigurable Computing **2016** (2016).

[4] J. Pilch and J. Długopolski, Journal of Computational Electronics **18**, 329 (2019).

[5] M. A. Nielsen and I. L. Chuang, Phys. Today **54**, 60 (2001).