Bachelor of Science in Electrical and Electronic Engineering

EEE 400 (January 2024): Thesis

Design, Tuning and Optimization of a 4-Qubit Superconducting Quantum Chip

Submitted by

Fahim Shahriar Anim

S201906178

Supervised by

Dr. Md. Saifur Rahman

Professor

Department of Electrical and Electronic Engineering

Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

March 2025

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "Design, Tuning and Optimization of a 4-Qubit Superconducting Quantum Chip", is the outcome of the investigation and research carried out by me under the supervision of Dr. Md. Saifur Rahman.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Fahim Shahriar Anim
S201906178

CERTIFICATION

This thesis titled, **"Design, Tuning and Optimization of a 4-Qubit Superconducting Quantum Chip"**, submitted by me has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Electrical and Electronic Engineering(EEE) in March 2025.

Group Members:

Fahim Shahriar Anim

Supervisor:

Dr. Md. Saifur Rahman
Professor
Department of Electrical and Electronic Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

At the very outset, I would like to express my gratitude to the Almighty Allah for bestowing upon me the strength, resilience, and determination to successfully complete this thesis within the given timeframe. I am deeply indebted to my thesis supervisor, Dr. Md. Saifur Rahman,Professor, Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology (BUET), for his invaluable guidance, continuous support, and insightful feedback, which have been instrumental in shaping the direction of this research.

I would like to extend my sincere appreciation to Dr. Sajid Muhaimin Chowdhury, my academic advisor, for his mentorship and unwavering support during my academic journey. His advice and encouragement have also played a crucial role in my growth as a researcher.

I extend my appreciation to the Qiskit Metal team for their extensive resources and tools, which greatly facilitated the design and simulation process of my superconducting quantum chip. Their open-source contributions have been vital in advancing research in this field. Additionally, I would like to thank Dr. Hiu Yung Wong, Associate Professor at San Jose State University, for his insightful YouTube playlist on superconducting quantum computing. His educational content provided a strong foundation and practical insight which enriched my understanding of the subject/matter.

Finally, I remain ever grateful to my parents and family, whose unwavering support and prayers have been instrumental in my journey. Along with them, I extend my appreciation to all individuals and communities who have assisted me in the successful completion of this thesis. Their encouragement and contributions have played a significant role to make this achievement possible.

Fahim Shahriar Anim

Dhaka
March 2025

Contents

List of Figures

List of Tables

ABSTRACT

Superconducting quantum computing is a leading platform for scalable quantum computation, leveraging low-loss superconducting circuits to achieve high coherence and controllability. It enables the realization of quantum algorithms for solving complex problems, with applications in cryptography, optimization, and materials science. In this work, two distinct 4-qubit superconducting quantum chips were designed and analyzed using *Qiskit Metal* and *Ansys Q3D and HFSS*. The first design includes two Transmon Pocket qubits with charge lines and two Transmon Cross qubits with flux lines, each coupled to two neighbors via dedicated resonators, with separate readout for all. The qubit frequencies, anharmonicities, and qubit-resonator coupling strengths and readout dispersive shifts were optimized by tuning geometric parameters and Josephson junction properties, using Lumped Oscillator Model (LOM) analysis. In addition, the coupling and readout resonators were tuned through eigenmode analysis to achieve the desired properties. A full-chip eigenmode simulation verified the expected frequency distribution in 12 modes (qubits, coupling resonators, and readout resonators). The surface participation ratio of the whole chip was also analyzed using Energy Participation Ratio (EPR) calculations to assess dielectric losses. To explore alternative connectivity schemes, a second 4-qubit chip design was developed, where all qubits were coupled to a central bus resonator instead of using individual coupling resonators. Although this design was not fully optimized, it serves as a comparative study to evaluate differences in qubit connectivity. Finally, examples of several practical applications of the 4-qubit chip have been demonstrated. The findings from this research provide insights into the impact of the geometry of components and their coupling architectures on quantum chip performance, forming future superconducting qubit layouts for improved gate operations and coherence.

Chapter 1

Introduction

Quantum computing has emerged as a rapidly advancing field with the potential to redefine computational paradigms across various domains. It harnesses the principles of quantum mechanics to process information. In this chapter, significance of quantum computing is discussed. Among different types of quantum computing mechanisms, superconducting quantum computing is one of the most promising approaches in the quantum computing landscape. This chapter overviews a review of superconducting quantum computers over the decades. Additionally, this chapter includes the motivation for the current work followed by the main objectives of this work.

1.1 Importance of Quantum Computing

Quantum computers hold a distinct advantage over classical counterparts in solving computationally challenging problems that are otherwise intractable. Unlike classical computers, which rely on binary bits (0 or 1) for processing information, quantum computers leverage quantum bits (qubits) that can exist in a superposition of states, offering exponential speedups for specific problem classes [1]. Furthermore, quantum entanglement, a phenomenon where qubits become interdependent regardless of distance—allows for highly correlated computations, significantly enhancing processing power beyond classical architectures [2]. Over the past few decades, quantum computation has transitioned from theoretical exploration to practical implementation, attracting increasing attention from researchers, industry, and investors. Early theoretical breakthroughs in the 1990s, such as Shor's algorithm for large integer factorization, quantum error correction, and quantum computers as universal quantum simulators, laid the foundation for this field. Shor's algorithm (1994) enables efficient factorization of large numbers, posing a direct threat to classical cryptographic security [3], while Grover's algorithm (1996) provides a quadratic speedup in searching unsorted databases [4]. Beyond these theoretical breakthroughs,

quantum computing has demonstrated significant potential in optimization problems [5], drug discovery [6], and materials science, where classical computers struggle due to the exponential growth of complexity. Since Feynman (1982) first proposed quantum simulators for modeling complex physical systems, the field has seen remarkable progress, evolving from fundamental research in quantum mechanics to the engineering of multi-qubit quantum systems capable of real-world computations [7, 8]. The rise of quantum engineering as a distinct discipline has further accelerated this transition, bridging quantum mechanics with engineering applications to develop scalable and fault-tolerant quantum devices [9, 10]. As a result, quantum computing is increasingly seen as a transformative technology that could drive innovation in diverse fields, from logistics and artificial intelligence to financial modeling and cryptography [11]. Additionally, while classical supercomputers require exponential time to simulate quantum systems accurately, quantum processors can solve these problems in polynomial time [12]. However, even though they have theoretical advantages, quantum computers still face significant challenges, including qubit decoherence, noise, and the need for advanced error correction techniques [8]. Despite these hurdles, ongoing advancements in quantum hardware and algorithms continue to push the field toward practical and real-world applications.

1.2 Role of Superconducting Qubits: A Review

Superconducting quantum computing (SQC) has emerged as one of the most promising and scalable platforms for quantum information processing. Unlike other quantum computing paradigms that encode quantum information in natural microscopic systems—such as trapped ions [13], quantum dots [14], and photonic qubits [15]—superconducting qubits are macroscopic in size and lithographically defined on semiconductor chips [16]. This unique property allows researchers to engineer qubits with tailored properties by designing superconducting circuits, often referred to as ”artificial atoms,” due to their atom-like discrete energy levels [17]. These artificial atoms leverage Josephson junctions as nonlinear inductive elements, which enable the creation of an anharmonic quantum oscillator necessary for quantum computation [18]. The rapid advancements in superconducting quantum circuits over the last two decades have transformed them from interesting physical devices to viable candidates for large-scale quantum computing. Arthur et al. [19] depicted superconducting qubits as the most mature and well-funded quantum computing technology, while other qubit platforms remains in early research phases as shown in Figure 1.1.

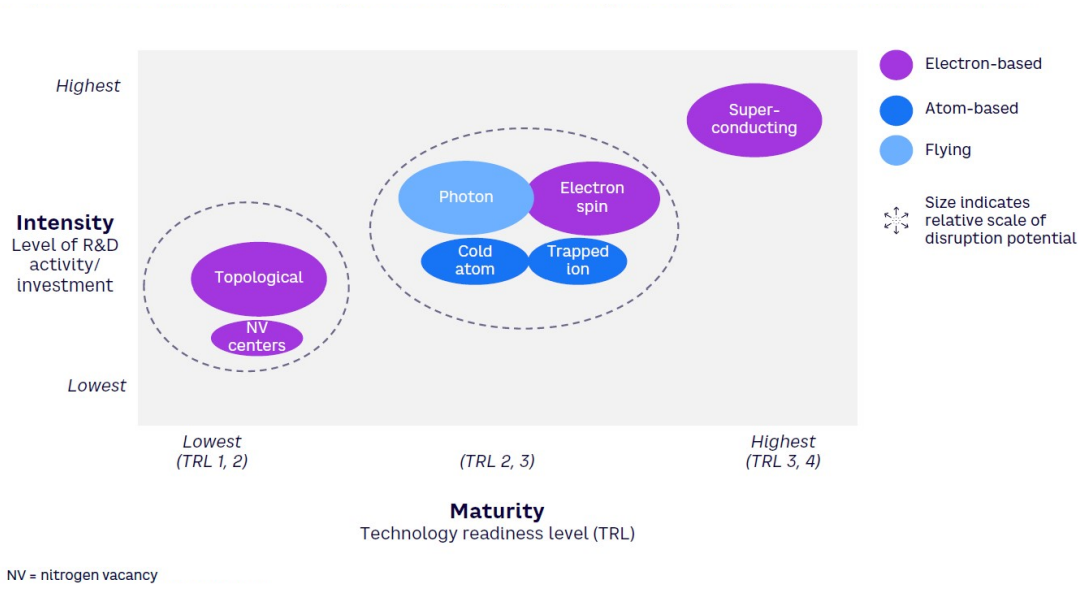


Figure 1.1: Core Qubit Technologies mapped by maturity, intensity and disruption potential. [Source: Arthur D. Little, Olivier Ezratty]

Among the various superconducting qubit designs, transmon qubits have become the most widely used due to their improved coherence times and reduced sensitivity to charge noise [20]. Since the first successful demonstration of superconducting charge qubits in 1999 [18], significant advancements have been made in extending coherence times, enhancing qubit connectivity, and refining error correction techniques. Transmon qubits are modified versions of Cooper-pair box qubits with increased shunt capacitance, improving coherence at the cost of reduced anharmonicity. Other designs include flux qubits, which rely on the quantum states of circulating currents, and fluxonium qubits, which offer long coherence times by utilizing an array of Josephson junctions [21]. Barends et al. [22] presented the 'Xmon' qubit in 2013, paving a way for easily connected designs of superconducting quantum processors. The design included a cross shaped qubit capacitor with coupling capacitances on each end, leading to control and readout lines and a nearby Josephson Junction with a frequency tunable SQUID loop. However, superconducting qubits still face notable challenges, including decoherence and susceptibility to environmental noise, necessitating cryogenic cooling to millikelvin temperatures for stable operation [23]. Their relatively short coherence times require extensive quantum error correction to achieve fault-tolerant quantum computing [24]. Additionally, scaling superconducting qubits in large quantum processors introduces complications such as increased crosstalk and the need for sophisticated control electronics, making scalability a persistent challenge [25].

Efficient quantum computing requires well-controlled qubit-qubit interactions, which are facilitated through capacitive coupling, inductive coupling, or resonator-mediated interactions. Direct capacitive coupling is commonly used in fixed-frequency transmon qubits, but tunable coupling schemes, such as gmon qubits [26] and parametric tunable couplers, allow for dynamic

control of interaction strengths. The use of bus resonators to mediate multi-qubit interactions has proven effective in scaling superconducting quantum processors [27].

One of the key advantages of superconducting qubits is their strong designability, allowing precise control over transition frequencies, anharmonicity, and coherence properties [28]. Unlike naturally occurring atomic systems, superconducting circuits provide a highly flexible and scalable architecture for quantum processors, which has led to significant industry investment from major technology companies such as Google, IBM, and Rigetti Computing [29]. The field saw a major breakthrough with Google's demonstration of quantum supremacy, where a 53-qubit superconducting processor named Sycamore executed a task in seconds that would take classical supercomputers thousands of years [8]. Recent advancements in superconducting quantum computing, such as Google's **Willow** processor with 105 qubits and IBM's cloud-based Quantum Platform, demonstrate rapid progress in scalability and error correction [30]. Microsoft's **Majorana 1** chip further enhances qubit stability using topological superconductors, advancing the field toward fault-tolerant quantum computing [31]. **Zuchongzhi 3.0**, revealed in **March 3, 2025**, is a 105-qubit superconducting quantum computer prototype that achieves high operational fidelities, with single-qubit gates, two-qubit gates, and readout fidelity at 99.90%, 99.62%, and 99.13%, respectively [32]. Amazon's **Ocelot**, released in **February, 2025**, represents an important step on the road to practical quantum computers, leveraging chip-scale integration of cat qubits to form a scalable, hardware-efficient architecture for quantum error correction [33].

Superconducting quantum computers compete with various quantum hardware platforms, each with distinct advantages and challenges. Trapped-ion systems, for example, offer long coherence times and high gate fidelities but suffer from slow gate speeds, whereas superconducting qubits operate on nanosecond timescales, enabling faster quantum circuits at the expense of shorter coherence times [13, 34]. Although, the superconducting architecture of quantum computing has made significant strides over the past decades, and much progress in coherence times has been credited to innovative fabrication developments [35, 36]. Photonic qubits, on the other hand, do not require cryogenic cooling and exhibit virtually infinite coherence times; however, implementing logic gates in photonic systems remains challenging due to their inherently probabilistic operations [15]. Another competing platform is neutral atom-based qubits, which provide scalability and long coherence times but still lag in maturity compared to superconducting technology [37]. Additionally, topological qubits, a theoretical approach being actively explored by Microsoft, hold promise for fault tolerance but have yet to be realized in practical quantum computing systems [38]. Despite these competing technologies, superconducting qubits remain one of the most advanced and scalable platforms currently available for quantum computation. According to the recent report *"A Portrait of the Global Patent Landscape in Quantum Technologies"* by European Quantum Industry Consortium (QuIC) in January, 2024 highlights that superconducting qubits currently represent two thirds of patent families related

to hardware quantum computing, with a strong lead from US companies, as shown in Figure 1.2.

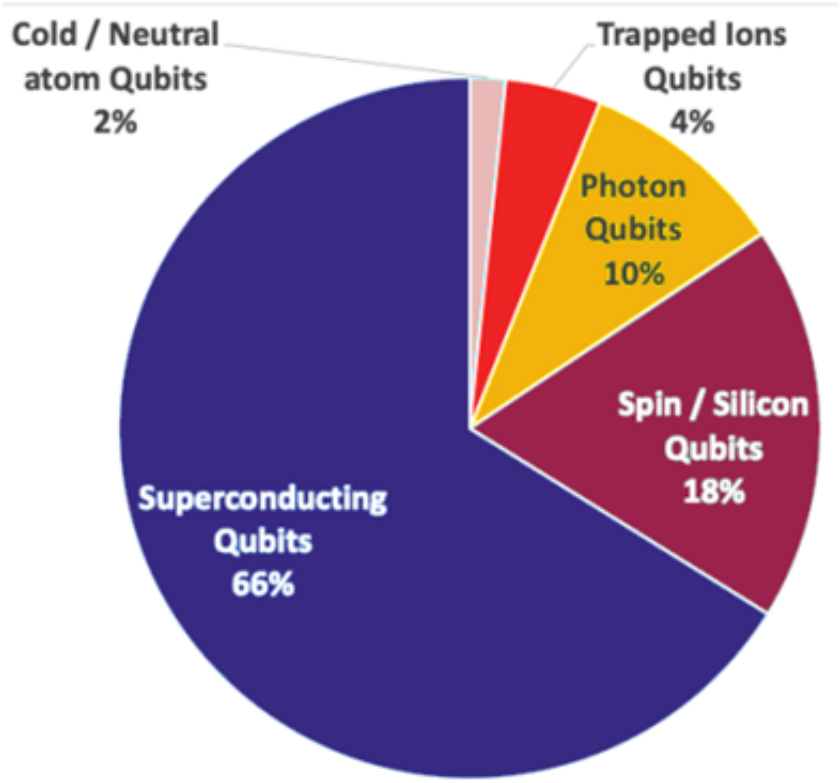


Figure 1.2: Number of patent families by hardware segment in quantum computing.

Overall, superconducting qubits are the basis of many efforts to build large scale quantum computers, and have enabled key demonstrations of quantum algorithms [39], quantum error correction [40–43], quantum many body physics, [44, 45] and quantum advantage in performing specific tasks [8]. Furthermore, they are a promising technology for quantum sensing, hybrid systems coupling different types of quantum emitters [46,47], and realizing large-scale quantum computing architectures [25].

1.3 Research Motivation for a 4-Qubit Chip

A 4-qubit superconducting quantum chip is a crucial stepping stone in quantum computing research, serving as both a fundamental building block and a practical testbed for scalable systems. It extends beyond simple 2-qubit circuits while remaining feasible in terms of design, simulation, and fabrication. As an experimental platform, it enables researchers to study multi-qubit interactions, entanglement, and gate fidelities, which are essential for improving coherence times and error rates before scaling to larger quantum processors. Many commer-

cial quantum computing companies, including IBM, Rigetti, and Google, have utilized smaller multi-qubit architectures to benchmark small-scale quantum systems and validate their methodologies before moving to larger-scale implementations. Figure 1.3 is one of IBM's earlier smaller superconducting device consisting 5 qubits.

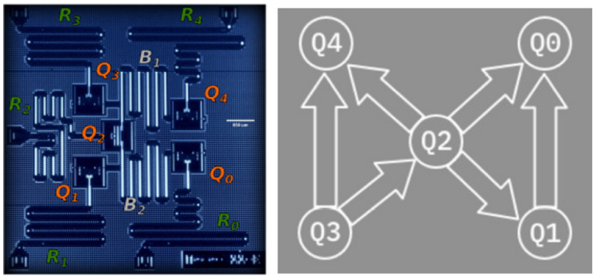


Figure 1.3: (a) IBM's 5-Qubit chip structure *ibmqx4*, (b) Connectivity diagram of *ibmqx4*.

Additionally, a 4-qubit system is large enough to execute fundamental quantum algorithms such as Grover's Search, the Quantum Fourier Transform (QFT), and Variational Quantum Eigensolver (VQE)—providing insights into quantum advantage. This chip size also facilitates research into qubit connectivity and coupling, allowing different topologies (e.g., linear, star, or fully connected) to be tested, optimizing gate execution and mitigating crosstalk effects. Furthermore, the simulation feasibility of 4-qubit systems using tools like Qiskit Metal and Ansys allows researchers to balance computational complexity with practical implementation. Given that early-stage quantum processors from leading companies began at this scale, a 4-qubit superconducting chip provides a vital framework for testing and optimizing quantum hardware before progressing toward fault-tolerant, large-scale quantum computation.

Furthermore, in recent works, physical qubits in superconducting systems have reached the point where errors are at or below the threshold, and networks of 4-9 superconducting qubits with individual control and readout have been used to show concepts of error correction [40,42, 48].

1.4 Objectives of the Current Work

The major objectives of this research are as follows-

- Demonstration of design procedure of a small multi-qubit (4-qubit) superconducting quantum chip.
- Overview of tuning mechanisms of the components of this chip.
- Demonstration of the effects of geometric or characteristic changes of the components on the desired chip parameters.

- Comparison between two different qubit-connectivity mechanism.
- Demonstration of probable real world applications using a small quantum chip.

1.5 Contributions of this Thesis

This work overviews an overview of the step-by-step design and tuning process of a small scale superconducting quantum chip. This work includes a brief comparison of two basic qubit-qubit connectivity or coupling schemes in a quantum chip. Also, the computational costs for tuning analysis of various components were demonstrated, along with some small scale applications.

1.6 Organization of this thesis

The first chapter gives a brief overview of the significance of quantum computing as a rapidly emerging technology, role of superconducting chips in realization of a practical quantum computer over the years and the motivation, as well as the objectives of this work. The second chapter includes a brief theoretical background of the foundation of quantum computing and superconducting quantum computing, the real word challenges for building a quantum computer, followed by the structure of a practical superconducting quantum computer. The third chapter depicts the simulation methodology, the designs of the chip and fabrication considerations. The fourth chapter includes tuning results of various components and comparison between two chips having different qubit connectivity and the final chapter includes some probable applications and scope for further optimization of this work.

Chapter 2

Theoretical Framework

In the previous chapter, the significance of quantum computing and revolution of superconducting quantum computing over last few decades was discussed. In this chapter, a brief theoretical aspect of superconducting qubits and superconducting quantum computer is discussed along with an overview of quantum computing.

2.1 Overview of Quantum Computing

In this section, an overview of fundamental properties of a qubit and operation of different single-qubit gates and multi-qubit gates is provided, which will help understand the design aspects and basic concept of the subject matter.

2.1.1 Properties of a Qubit

In a classical computer, a bit is represented by a physical quantity with two states to represent 0 and 1. It can be the voltage in a circuit (e.g. 5 V for 1 and 0 V for 0), magnetization direction in a magnetic drive (e.g. up for 1 and down for 0) or charge density in the charge storage layer in the flash memory (e.g. low for 1 and high for 0). In a quantum computer, the information is represented by qubits. Like the classical bit, each qubit still has two distinct states which are orthogonal to each other because they cannot exist with 100 % certainty at the same time. They are known as the **basis vectors**. The basis vectors can be written as-

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.1}$$

In addition to the two distinct states, a qubit may also have many other states which are the

superposition of the two distinct states (or the basis states):

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.2}$$

where $|0\rangle$ and $|1\rangle$ are the computational basis states, and α and β are complex probability amplitudes satisfying $|\alpha|^2 + |\beta|^2 = 1$. This superposition allows quantum computers to process multiple computations in parallel, contributing to their potential computational advantage over classical systems.

A single qubit state can be represented geometrically in three dimensional space using **Bloch Sphere** representation. It provides an intuitive way to visualize the superposition of quantum states and the effects of quantum gates on a qubit. A general qubit state can be written as:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{2.3}$$

where θ (polar angle) and ϕ (azimuthal angle) are real numbers that define the position on the sphere. The term $e^{i\phi}$ represents a phase factor that captures quantum phase information as shown in Figure 2.1.

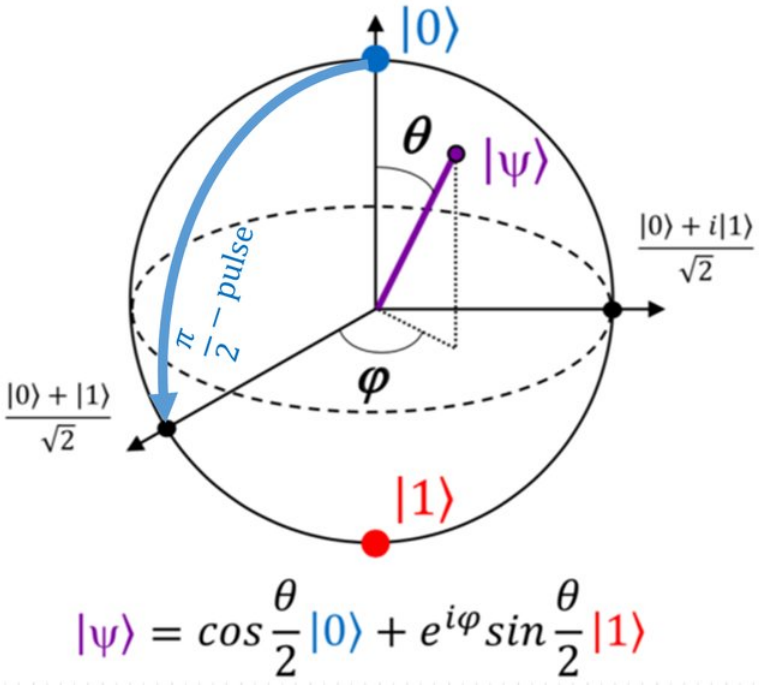


Figure 2.1: Bloch Sphere representation of Qubits.

Quantum measurements are probabilistic. When a qubit is measured, the state is collapsed to one of the basis states associated with the measurement. In general, the probability that a state

$|\psi\rangle$ will be found in the basis state $|\alpha\rangle$ when measured is given by

$$Pr(|\alpha\rangle) = |\langle\psi|\alpha\rangle|^2. \tag{2.4}$$

This is known as the ***Born Rule***.

Apart from superposition, some other properties of a qubit are-

- **No Cloning:** Unlike classical bits, quantum states cannot be perfectly copied due to the **no-cloning theorem**. While quantum states can be transferred using **entanglement** (teleportation), this process destroys the original state rather than duplicating it.
- **Reversibility of Gates:** Quantum gates are inherently **reversible**, meaning that applying the correct inverse operation can recover the original state. This contrasts with many classical logic gates, which lose information when processing multiple inputs.
- **Entanglement:** Entangled qubits exhibit strong correlations such that measuring one instantly determines the state of the other, regardless of distance. However, since measurement outcomes are probabilistic, **no information is transmitted faster than light**, preserving the principles of relativity.

Table **??** shows the fundamental differences between classical bits and qubits.

Table 2.1: Comparison between *Classical Bits* and *Quantum bits*

Feature	**Classical Bits**	**Quantum Bits (Qubits)**
Representation	0 or 1 (binary voltage levels)	Superposition of 0 and 1 (Basis States)
Processing Power	Processes one n-bit state at a time	Can process 2^n n-bit states at once (Parallelism)
Logical Operations	Based on Boolean Logic Gates (Non-Reversible)	Uses Quantum Gates (Unitary, Reversible)
Error Sensitivity	Resistant to noise and stable at high temperatures	Highly sensitive to decoherence and environmental noise
Measurement	Readout is deterministic and repeatable	Measurement collapses the state probabilistically
Copying Information	Can be copied without restriction	Subject to the No-Cloning theorem, cannot be copied exactly
Hardware	Uses Transistors in Silicon chips	Requires Superconducting circuits, Trapped ions, Neutral atoms or other quantum systems

2.1.2 Quantum Gates

Quantum gates are fundamental operations that manipulate qubits by applying unitary transformations, preserving quantum coherence and enabling reversible computation. Figure 2.2 demonstrates examples of some single qubit gates.

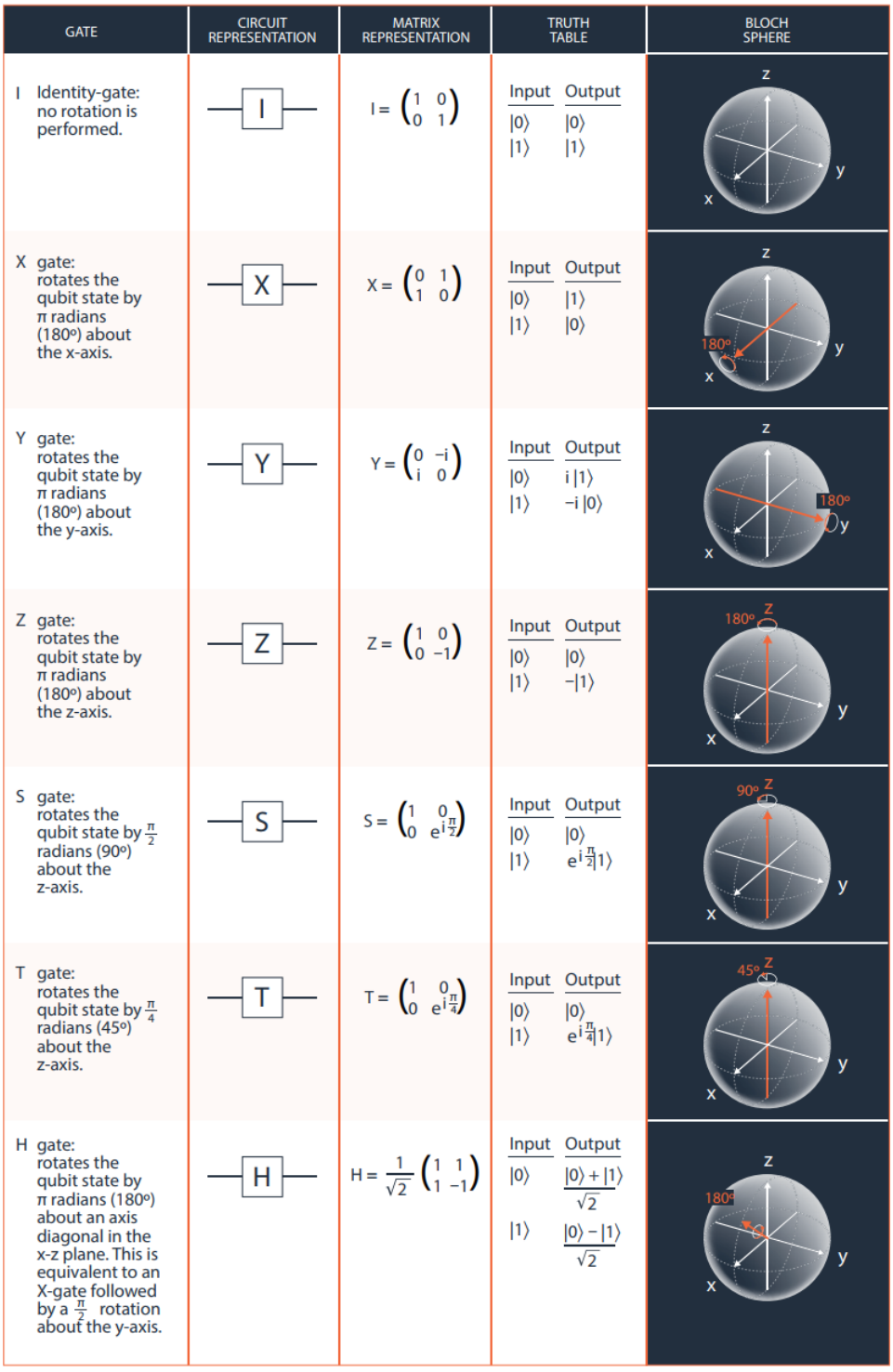


Figure 2.2: Examples of Single Qubit Gates.

In quantum computing, any **single-qubit unitary operation** can be expressed in terms of the *U3* gate, making it a universal single-qubit gate.

$$U3(\theta, \phi, \lambda) \equiv \begin{bmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} & e^{i(\phi+\lambda)}\cos\frac{\theta}{2} \end{bmatrix}. \tag{2.5}$$

The effect of a single qubit gate is to change α and β into a new mixture α' and β', which can be written as a matrix equation-

$$\begin{bmatrix} \alpha' \\ \beta' \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{2.6}$$

where,

$$|\psi'\rangle \equiv U|\psi\rangle.$$

where, the matrix U is referred to as unitary operator.

Examples of single-qubit gates include the **Pauli gates** (X, Y, Z), the **Hadamard gate** (H), and the **Phase gate** (S, T).

Multi-qubit gates include the **CNOT (Controlled-NOT) gate**, the **Toffoli gate (CCNOT)**, and the **SWAP gate**. Figure 2.3 demonstrates some examples of multi-qubit gates.

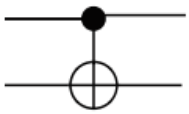
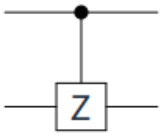
GATE	CIRCUIT REPRESENTATION	MATRIX REPRESENTATION	TRUTH TABLE
Controlled-NOT gate: apply an X-gate to the target qubit if the control qubit is in state $\|1\rangle$		$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	Input: $\|00\rangle$, $\|01\rangle$, $\|10\rangle$, $\|11\rangle$ — Output: $\|00\rangle$, $\|01\rangle$, $\|11\rangle$, $\|10\rangle$
Controlled-phase gate: apply a Z-gate to the target qubit if the control qubit is in state $\|1\rangle$		$\text{CPHASE} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$	Input: $\|00\rangle$, $\|01\rangle$, $\|10\rangle$, $\|11\rangle$ — Output: $\|00\rangle$, $\|01\rangle$, $\|10\rangle$, $-\|11\rangle$

Figure 2.3: Examples of Multi Qubit Gates.

In superconducting quantum circuits, qubit interactions are often mediated through capacitive or inductive coupling, making **SWAP-based gates** more natural than **CNOT gates**. The **iSWAP gate** is a crucial two-qubit gate in superconducting quantum computing, where it facilitates **state**

exchange between qubits while introducing a phase factor. Unlike the **CNOT gate**, which is commonly used for quantum logic operations, the iSWAP gate naturally fits superconducting architectures due to the way qubits interact via capacitive or inductive coupling. Mathematically, it is represented as:

$$\text{iSWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.7}$$

This gate can be implemented using resonant qubit interactions or tunable couplers in superconducting quantum circuits, allowing for high-fidelity **entanglement generation**. Additionally, its variant, the $\sqrt{\textbf{iSWAP}}$ **gate**, is frequently used in **universal gate sets**, and the **fSim gate** extends iSWAP with a tunable phase term, enabling efficient quantum simulations.

2.2 Superconducting Quantum Circuits

This section describes the basics of superconductivity, theory of Josephson Junctions, the building block of a superconducting quantum chip and different types of superconducting qubits.

2.2.1 Theory of Superconductivity

Superconductors exhibit zero resistance to DC currents and display unique properties below a critical temperature, distinguishing them from ordinary conductors. Below a **critical temperature**, conduction electrons in certain materials form **Cooper pairs** due to an attractive interaction, lowering the system's ground state energy. Figure 2.4 shows the formation of Cooper pair in an ion lattice.

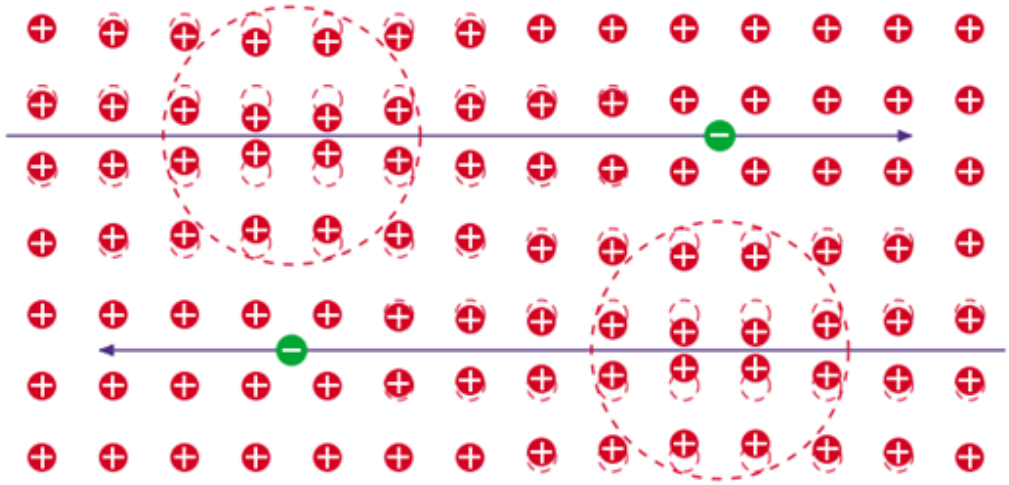


Figure 2.4: The delayed response of heavy ions to a rapidly moving electron creates a local positive charge concentration, which subsequently attracts another electron, effectively leading to an electron-phonon interaction that mediates **Cooper pair formation**.

The **superconducting energy gap** (Δ) prevents small perturbations from disrupting the Cooper pairs, leading to effects like persistent supercurrents and magnetic field expulsion **(Meissner effect)**, unless the applied energy exceeds 2Δ. Other properties include macroscopic quantum coherence [49], and distinctive behavior in weakly coupled junctions (Josephson effect) [50].

2.2.2 From LC Resonator to Josephson Junction

In quantum mechanics, the **Hamiltonian** of a simple harmonic oscillator is given by:

$$H = \frac{p^2}{2m} + \frac{1}{2}kx^2. \tag{2.8}$$

The Hamiltonian is a mathematical operator that represents the total energy (sum of kinetic and potential energy) of a quantum system and governs its time evolution via the Schrödinger equation.

Using the **ladder operator formalism**, we define the **creation** and **annihilation** operators [51]:

$$a^\dagger = \frac{1}{\sqrt{2\hbar m\omega}}\left(-ip + m\omega x\right), \quad a = \frac{1}{\sqrt{2\hbar m\omega}}\left(ip + m\omega x\right). \tag{2.9}$$

Rewriting the **Hamiltonian** in terms of these operators:

$$H = \hbar\omega\left(a^\dagger a + \frac{1}{2}\right). \tag{2.10}$$

The **energy eigenvalues** are given by:

$$E_n = \hbar\omega\left(n + \frac{1}{2}\right), \quad n = 0, 1, 2, \ldots \tag{2.11}$$

where the **zero-point energy** is $E_0 = \frac{1}{2}\hbar\omega$, indicating that even in the lowest energy state, the oscillator has a nonzero energy due to quantum fluctuations [51].

A **Linear LC resonator** consists of an **inductor** (L) and a **capacitor** (C), forming an electrical circuit that behaves like a harmonic oscillator. The total energy in the circuit is the sum of **electric energy** stored in the capacitor and **magnetic energy** stored in the inductor:

$$H = \frac{q^2}{2C} + \frac{\Phi^2}{2L} \tag{2.12}$$

where q is the charge on the capacitor and Φ is the magnetic flux. The system oscillates with a natural **resonant frequency**:

$$\omega_r = \frac{1}{\sqrt{LC}}. \tag{2.13}$$

In quantum mechanics, the LC circuit can be treated analogously to the quantum harmonic oscillator, with **quantized energy levels** given by equation (2.11).

Although a bare LC circuit cannot function as a qubit due to its harmonic energy spectrum, it still plays a fundamental role in superconducting qubits by serving as a building block for quantum state manipulation and coupling mechanisms. [52, 53].

Now an important question arises: **Can a harmonic oscillator *(like LC resonator)* circuit act as a qubit?** The answer is **NO!** For a system to function as a qubit, it must have exactly two states. From figure 2.5 collected from [54], a harmonic oscillator has an infinite number of equally spaced energy levels, making it unsuitable for qubit implementation. If we attempt to use a harmonic oscillator as a qubit, initializing it in the **ground state** and applying a π-pulse would move it to the **first excited state**. However, a second π-pulse intended to return it to the ground state could just as easily excite it to the $n = 2$ level, due to the uniform energy spacing and lack of **anharmonicity**. Hence, any physical systems with more than two states can function as qubits, provided that their energy level spacings are sufficiently different, allowing selective excitation of only two levels using appropriately tuned pulse frequencies. The challenge with the harmonic oscillator is its **equally spaced energy levels**, a direct result of its quadratic potential. By introducing **anharmonicity**—a deviation from the quadratic potential—the level spacing becomes non-uniform, enabling selective two-state manipulation necessary for qubit operation.

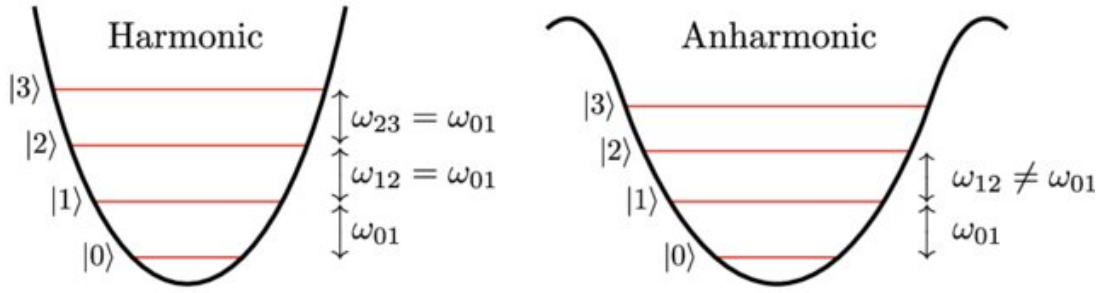


Figure 2.5: Harmonic and anharmonic systems and their suitability as qubits.In the quadratic potential of a harmonic system, energy levels are equally spaced ($\omega_{j,j+1} = \omega_{01}$), allowing population transfer beyond $|0\rangle$ and $|1\rangle$, whereas in an anharmonic system (e.g., Josephson junctions), unequal level spacing ($\omega_{01} \neq \omega_{12}$) ensures selective two-level transitions, forming a qubit.

To introduce the required nonlinearity to modify the harmonic potential, the Josephson junction – a nonlinear, dissipation-less circuit element that forms the backbone in superconducting circuits is mostly used [50, 55]. By replacing the linear inductor of the QHO with a Josephson junction, playing the role of a nonlinear inductor, we can modify the functional form of the potential energy.

The Josephson Junction consists of two superconducting regions separated by a non superconducting layer—usually an insulator. A typical junction is fabricated from aluminum films with thicknesses in the range of 35–85 nm, where the native oxide of aluminum serves as the insulating barrier. The key quantum mechanical effect underlying the Josephson junction's behavior is Cooper pair tunneling across the non-superconducting barrier, leading to the **Josephson equations**: **Current-Phase Relation(Josephson Super Current Equation):**

$$I = I_c \sin \phi \tag{2.14}$$

Voltage-Phase Relation (Josephson Voltage Equation):

$$V = \frac{\hbar}{2e} \frac{d\phi}{dt}. \tag{2.15}$$

where, I_c is the **critical current** of the junction, and ϕ is the **phase difference** across the junction. A Josephson junction is shown in Figure 2.6.

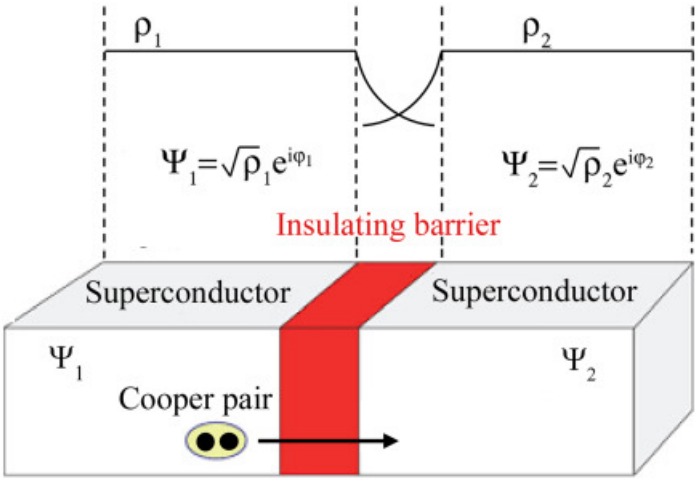


Figure 2.6: A Josephson junction, where φ represents the phase.

The Josephson junction therefore looks like a non-linear inductor; i.e., an inductor whose value depends on the current through it. The effective non-linear inductance of a Josephson Junction can be represented as -

$$L_{\text{eff}} = \frac{\Phi_0}{2\pi I_c \sqrt{1-(I/I_c)^2}}. \tag{2.16}$$

The potential energy stored in a Josephson junction is given by:

$$U(\phi) = -E_J \cos\phi. \tag{2.17}$$

where E_j is the **Josephson energy**, which can be written as:

$$E_J = \frac{\hbar I_c}{2e}. \tag{2.18}$$

A **SQUID (Superconducting Quantum Interference Device)** consists of two Josephson junctions forming a loop, allowing control of the effective Josephson energy via an external magnetic flux Φ. The **effective Josephson energy** of a SQUID is:

$$E_{J,\text{eff}} = E_J \cos\left(\frac{\pi\Phi}{\Phi_0}\right). \tag{2.19}$$

By tuning the external magnetic flux Φ, the Josephson energy can be dynamically controlled, enabling tunable qubits and coupling elements.

There are sevel types of superconducting qubits incorporating the Josephson Junction or a SQUID. A few are discussed below:

Chage Qubit

The **charge qubit**, also known as the **Cooper-Pair Box (CPB)**, consists of a small superconducting island connected to a superconducting lead via a Josephson junction. The **charging energy** E_C dominates, making the system highly sensitive to the number of Cooper pairs. The Hamiltonian is given by:

$$H = 4E_C(n - n_g)^2 - E_J \cos\phi. \tag{2.20}$$

where: n is the number of Cooper pairs on the island, $n_g = C_g V_g/2e$ is the **offset charge** controlled by an external gate voltage.

Charge qubits operate in the **strong charging regime**, where $E_C \gg E_J$. However, they suffer from **charge noise**, which severely limits coherence times.

Flux Qubit

The **flux qubit** consists of a superconducting loop interrupted by one or more Josephson junctions. The **phase degree of freedom** is used to encode quantum information, and the qubit is operated in the **strong Josephson regime** ($E_J \gg E_C$). The Hamiltonian of a flux qubit is:

$$H = -\frac{1}{2}\Delta\sigma_x - \frac{1}{2}\epsilon\sigma_z, \tag{2.21}$$

where, Δ is the **tunneling energy** between the two persistent current states and $\epsilon = 2I_p(\Phi - \Phi_0/2)$ is the **energy bias** due to an external magnetic flux Φ.

Flux qubits are **resistant to charge noise** but suffer from **flux noise**, which can limit their coherence times.

Phase Qubit

A **phase qubit** is similar to a flux qubit but operates in the **phase regime**, where the Josephson energy E_J dominates over the charging energy E_C ($E_J \gg E_C$). It is based on a **current-biased Josephson junction**. The Hamiltonian of a phase qubit is given by:

$$H = \frac{Q^2}{2C} - E_J \cos\phi - I_{\text{bias}}\Phi_0\frac{\phi}{2\pi} \tag{2.22}$$

where I_{bias} is the external bias current.

Phase qubits were among the earliest superconducting qubits but have largely been **superseded**

by transmon qubits due to their longer coherence times.

Transmon Qubit: Enhancing Charge Qubits

The **transmon qubit** is an improved version of the charge qubit, where a **large shunting capacitance** is added to suppress charge noise. It operates in the **weak charging regime** ($E_J \gg E_C$), reducing sensitivity to charge fluctuations. The energy levels are given by:

$$\hbar\omega_q \approx \sqrt{8E_C E_J} - E_C. \tag{2.23}$$

A key feature of transmon qubits is their **anharmonicity**, which allows selective two-level operation. The anharmonicity is defined as:

$$\alpha = E_{12} - E_{01} \approx -E_C. \tag{2.24}$$

The resonant frequency ω_0 of a transmon qubit in terms of the charging energy E_C and Josephson energy E_J is given by:

$$\omega_0 = \frac{\sqrt{8E_C E_J}}{\hbar} \tag{2.25}$$

where ω_0 is the resonant frequency of the transmon qubit, E_C is the charging energy, given by $E_C = \frac{e^2}{2C}$, where C is the total capacitance, E_J is the Josephson energy, related to the critical current I_c by $E_J = \frac{\hbar I_c}{2e}$.

Transmon qubits are widely used due to their **long coherence times**, ease of fabrication, and scalability. However, they require **stronger microwave control pulses** for gate operations due to their reduced anharmonicity.

Fluxonium Qubit

The **Fluxonium qubit** is a superconducting qubit designed to mitigate the limitations of charge and flux qubits by incorporating a large inductance. It consists of a Josephson junction shunted by a large superconducting inductance, typically implemented as a **Josephson junction array**. The large inductance suppresses phase fluctuations while preserving strong anharmonicity, making Fluxonium a promising candidate for long-lived quantum states. The Hamiltonian of a Fluxonium qubit is given by:

$$H = 4E_C\hat{n}^2 - E_J\cos\hat{\phi} + \frac{1}{2}E_L\hat{\phi}^2. \tag{2.26}$$

where $E_C = \frac{e^2}{2C}$ is the **charging energy**, $E_J = \frac{\hbar I_c}{2e}$ is the **Josephson energy**, $E_L = \frac{\Phi_0^2}{4\pi^2 L}$ is the **inductive energy**, determined by the large inductance L, $\hat{n}$ is the **Cooper pair number operator** and $\hat{\phi}$ is the **superconducting phase difference** across the junction. Fluxonium operates in the **strong inductive regime**, characterized by $E_L \sim E_C \ll E_J$.

This distinguishes Fluxonium from transmon and flux qubits. The large inductance leads to a highly anharmonic energy spectrum, enabling high-coherence qubits with reduced charge noise and long relaxation times. Fluxonium qubits are increasingly being considered for near-term quantum processors due to their enhanced coherence properties and compatibility with existing superconducting circuit architectures. Figure 2.7 shows a Charge Qubit and a Transmon Qubit.

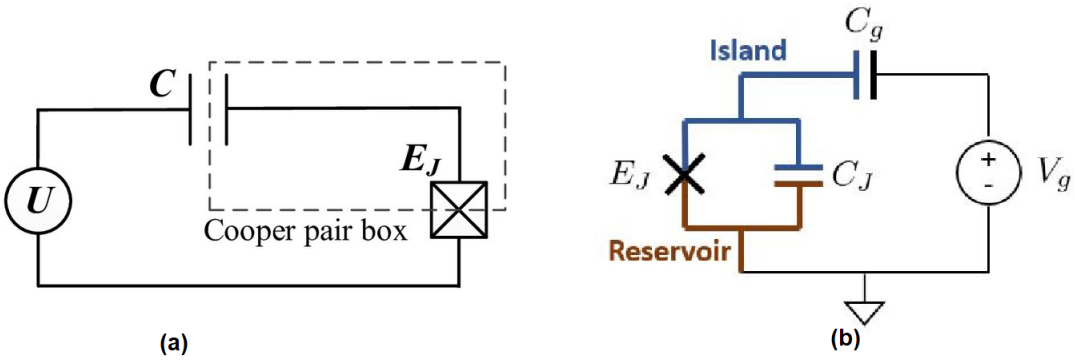


Figure 2.7: (a) Charge Qubit, (b) Transmon Qubit

There are also some **tunable Qubits** like **Xmon**, **Gmon**, **C-Shunt Flux Qubit** which provides better tunability and connectivity.

The summary of various superconducting qubit's operation regime is listed in the table 2.2.

Table 2.2: Operation Regimes and Value Ranges of Superconducting Qubits

Qubit Type	**Operation Regime**	**Typical Value Range**
Charge Qubit	Charge Regime	$E_J/E_C \ll 1$
Flux Qubit	Flux Regime	$E_J/E_C \gg 1$
Transmon Qubit	Weak Charge Dispersion	$E_J/E_C \approx 50 - 100$
Fluxonium Qubit	Hybrid Regime	$E_J \sim E_C \sim E_L$

Illustration of various superconducting qubits operating at various regimes is shown in Figure 2.8.

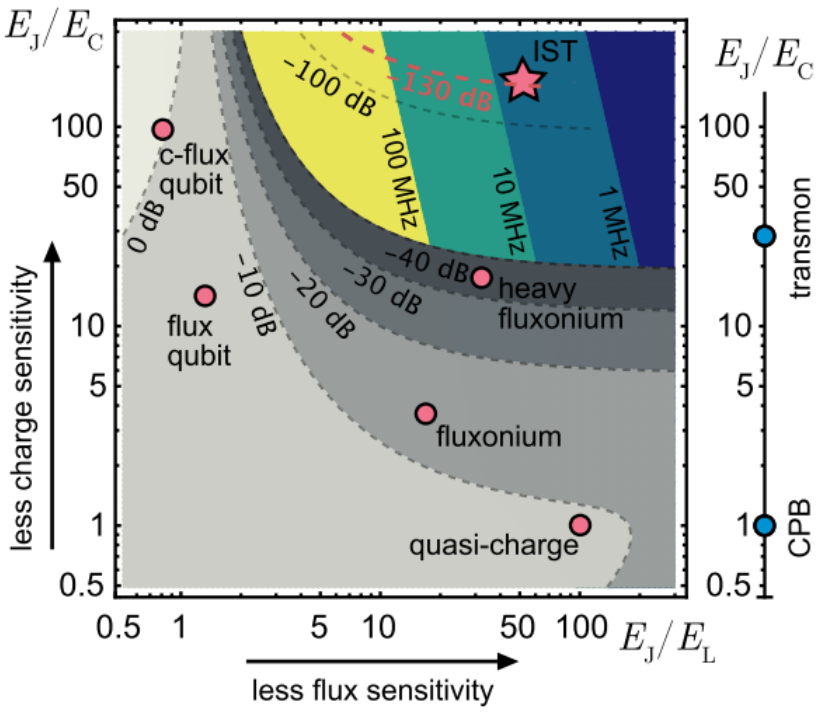


Figure 2.8: Classification Scheme for Superconducting Qubits (image extracted from [56]).

2.2.3 Circuit QED

Circuit Quantum Electrodynamics (cQED) describes the interaction between superconducting qubits and microwave resonators, enabling strong light-matter interactions in the microwave regime. This architecture is crucial for quantum computing with superconducting circuits.

Coupling of a Qubit to a Microwave Resonator

A superconducting qubit, such as a transmon, is coupled to a microwave resonator in cQED. The interaction is described by the Jaynes–Cummings Hamiltonian:

$$\hat{H} = \frac{\hbar\omega_q}{2}\hat{\sigma}_z + \hbar\omega_r\hat{a}^\dagger\hat{a} + \hbar g(\hat{a}\hat{\sigma}_+ + \hat{a}^\dagger\hat{\sigma}_-), \tag{2.27}$$

where, ω_q is the qubit frequency, ω_r is the resonator frequency, g is the qubit-resonator coupling strength, $\hat{a}^\dagger, \hat{a}$ are the creation and annihilation operators of the resonator and $\hat{\sigma}_z, \hat{\sigma}_+, \hat{\sigma}_-$ are the Pauli operators for the qubit.

This Hamiltonian describes the *coherent exchange of excitations* between the qubit and the resonator, known as **vacuum Rabi oscillations**. Different coupling mechanisms exist for qubit-resonator interaction:

- **Capacitive Coupling:** Capacitive coupling involves a shared capacitor between the qubit and the resonator. The coupling strength g depends on the capacitance values:

$$g \propto C_{qr}\sqrt{\frac{\omega_q\omega_r}{C_qC_r}}, \tag{2.28}$$

  where C_{qr} is the coupling capacitance, and C_q, C_r are the qubit and resonator capacitances.

- **Inductive Coupling:** For flux qubits and fluxonium qubits, coupling is achieved via a mutual inductance:

$$g \propto \frac{M}{\sqrt{L_qL_r}}, \tag{2.29}$$

  where M is the mutual inductance, and L_q, L_r are the qubit and resonator inductances.

The interaction strength g defines different coupling regimes:

- **Weak coupling:** $g \ll \kappa, \gamma$, where κ is the resonator decay rate, and γ is the qubit dephasing rate.
- **Strong coupling:** $g > \kappa, \gamma$, allowing coherent energy exchange.
- **Ultrastrong coupling:** $g \sim \omega_q, \omega_r$, where the rotating wave approximation (RWA) no longer holds.

Additionally, tunable couplers enable dynamic control over qubit interactions by adjusting the coupling strength between qubits or between qubits and resonators, allowing for flexible gate operations, suppression of crosstalk, and optimized quantum computing performance.

Qubit Readout

One of the key advantages of cQED is **non-demolition qubit readout** via dispersive coupling. In the **dispersive limit**, where the qubit-resonator detuning is large ($\Delta = \omega_q - \omega_r \gg g$), the system Hamiltonian can be approximated as:

$$\hat{H}_{\text{disp}} = \frac{\hbar}{2}(\omega_q + \chi\hat{a}^\dagger\hat{a})\hat{\sigma}_z + \hbar\omega_r\hat{a}^\dagger\hat{a}, \tag{2.30}$$

where χ is the **dispersive shift**, given by:

$$\chi = \frac{g^2}{\Delta}. \tag{2.31}$$

Figure 2.9 shows the dispersive shift in frequency of the readout resonator depending on qubit state.

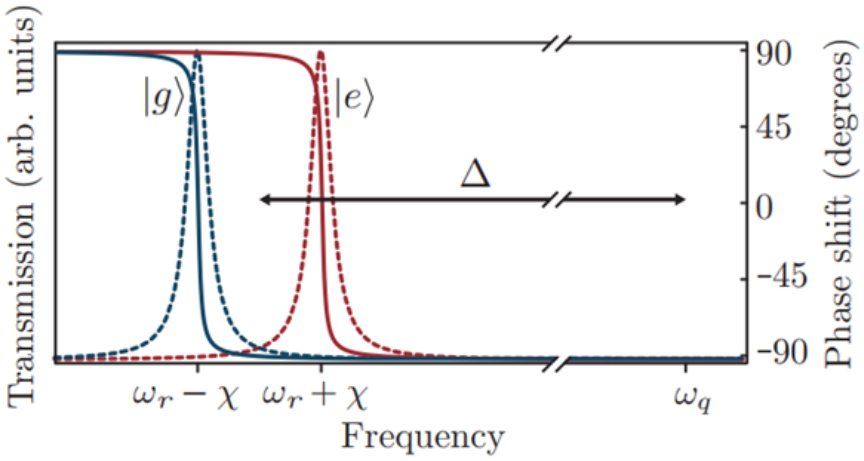


Figure 2.9: Readout resonator's dispersive shift depending on qubit state

The fidelity of dispersive readout depends on: Dispersive shift χ (larger χ improves contrast), cavity decay rate κ (too large κ leads to information loss), and qubit dephasing γ (dephasing should be minimized). For high-fidelity readout:

$$\frac{\chi}{\kappa} > 1. \tag{2.32}$$

The qubit readout process is essential in quantum computing for determining the quantum state of a qubit. Typically, the qubit is coupled to a resonator, which interacts with the qubit's quantum state. A microwave signal is sent through the resonator, and the interaction modifies the outgoing signal's properties, such as its amplitude or phase. By carefully analyzing these changes, the state of the qubit, whether $|0\rangle$ or $|1\rangle$ (or even a superposition), can be inferred. The measurement process is enumerated in a flow diagram shown in Figure 2.10.

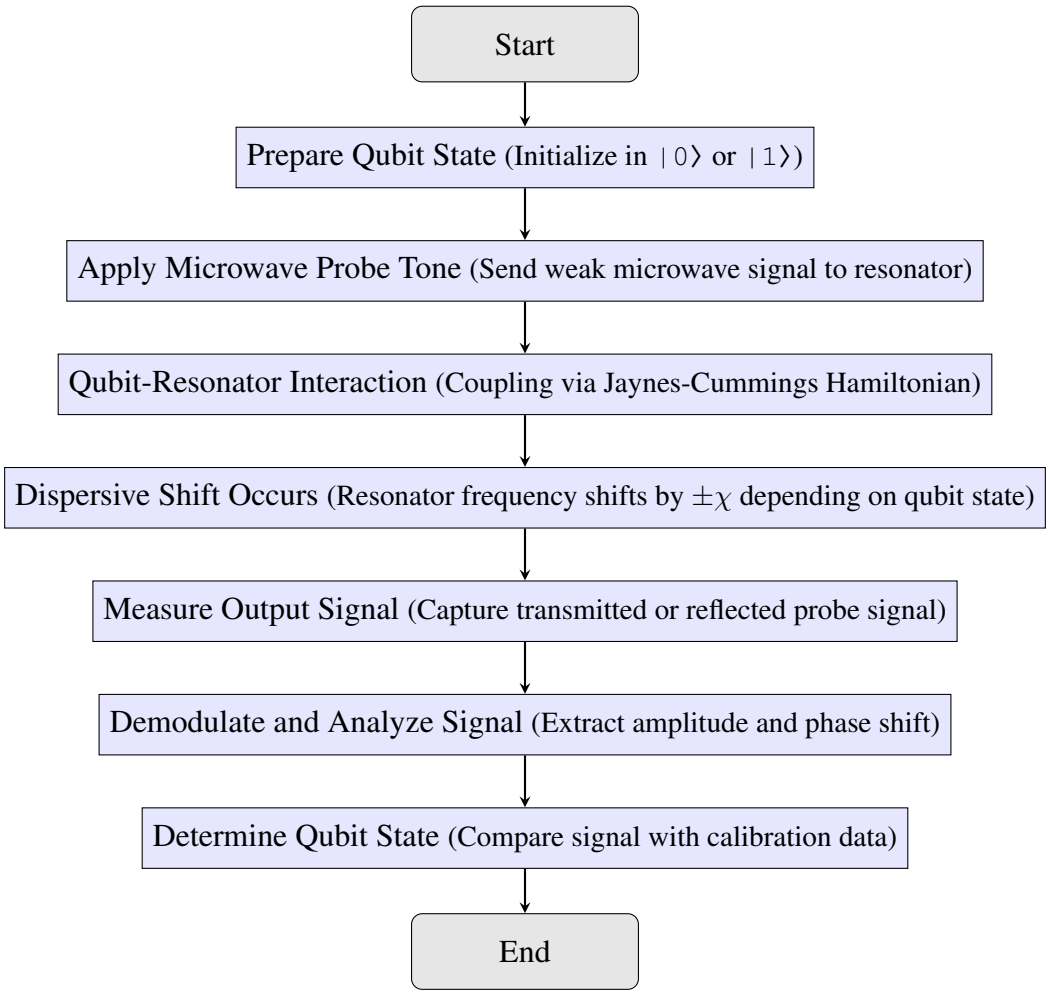


Figure 2.10: Qubit Readout and Measurement Process

2.2.4 Challenges

Scalability Limitation

Scalability is a major challenge in superconducting quantum circuits (SQC), as increasing the number of qubits leads to issues such as *crosstalk, frequency crowding, and wiring complexity*. As qubit numbers grow, unwanted interactions between nearby qubits and control lines introduce *spurious couplings*, reducing gate fidelity [8]. Additionally, superconducting qubits operate within a limited frequency range, making *frequency crowding* a critical problem where qubits with similar frequencies suffer from unwanted resonances and leakage errors [57]. The complexity of interconnecting a large number of qubits with *microwave control lines and cryogenic wiring* further complicates scalability, requiring advanced multiplexing techniques and novel architectures such as *modular quantum processors* and *chip-to-chip coupling strategies* to support large-scale quantum computing [58].

Control Electronics Complexity

Superconducting qubits rely on *precisely shaped microwave pulses* for gate operations, requiring sophisticated *high-speed, low-noise control electronics*. The need for *high-fidelity waveform generation, low-latency feedback loops, and scalable microwave routing* poses a major challenge, as current control hardware is often *bulky, power-hungry, and difficult to scale* to thousands of qubits [59]. Furthermore, operating qubits at *millikelvin temperatures* necessitates cryogenic-compatible control systems, pushing the development of *cryogenic electronics and on-chip control architectures* [60]. Innovations such as *high-density microwave signal routing, cryo-CMOS technology, and digital-to-analog converters (DACs) optimized for quantum systems* are being explored to overcome these challenges and enable practical large-scale superconducting quantum processors [61].

Qubit Decoherence

Qubit decoherence in superconducting quantum circuits arises from energy relaxation (T_1) and dephasing (T_2) mechanisms, ultimately limiting quantum coherence and computational fidelity.

- **Dielectric Loss:** Energy dissipation due to two-level system (TLS) defects in lossy interfaces of superconducting materials, leading to reduced qubit relaxation time (T_1). High-purity materials, surface passivation, and reduction of lossy dielectric participation enhance coherence.
- **Quasiparticle Poisoning:** Excess quasiparticles tunnel into Josephson junctions, causing parity fluctuations and increased energy relaxation [62]. Implementing quasiparticle traps and infrared shielding reduces the generation of non-equilibrium quasiparticles.
- **Flux Noise:** Magnetic fluctuations in superconducting loops introduce $1/f$ noise, leading to qubit frequency instability and dephasing [20]. Flux bias stabilization, improved fabrication techniques, and dynamical decoupling suppress flux noise effects [63].
- **Photon-Induced Dephasing:** Stray infrared and microwave photons excite qubits, leading to fluctuating energy levels and decoherence. Infrared shielding, cryogenic filtering, and improved thermalization help minimize photon-induced dephasing.
- **Charge Noise:** Trapped charge fluctuations in dielectrics cause shifts in qubit energy levels, particularly in charge-sensitive superconducting qubits. Using transmon qubits with reduced charge sensitivity and optimizing substrate cleanliness minimize charge noise [64].

- **Purcell Decay:** Qubit relaxation is accelerated by coupling to a lossy readout resonator, limiting excited-state lifetime [65]. Increasing qubit-resonator detuning, implementing Purcell filters, and using tunable couplers reduce unwanted energy loss [66].

2.3 Structure of a Superconducting Quantum Computer

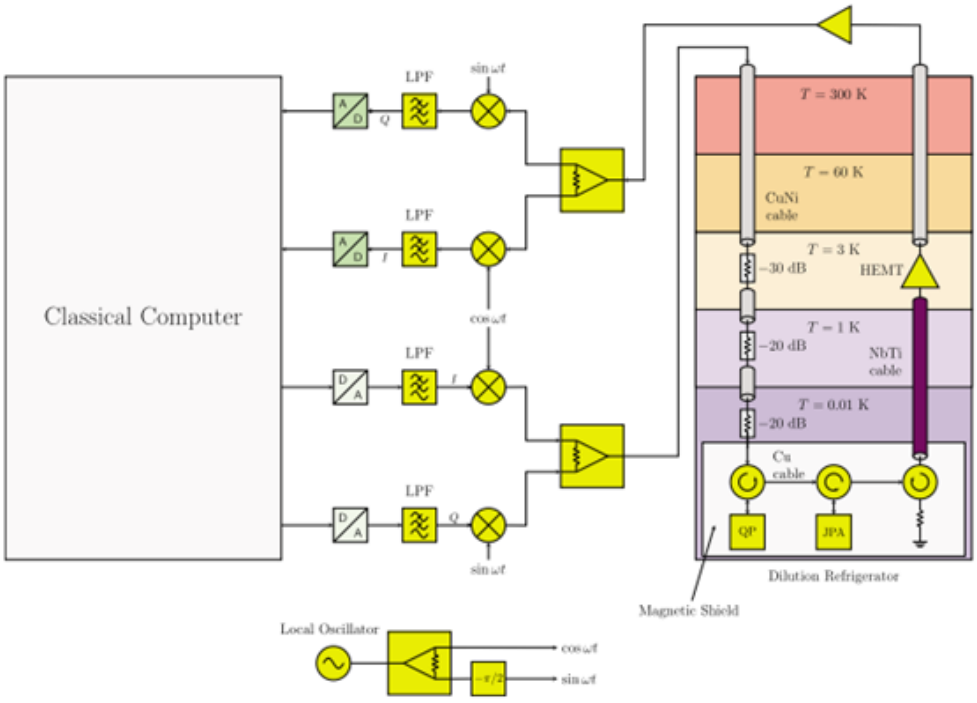


Figure 2.11: Structure of a Superconducting Quantum Computer

This system diagram in figure 2.11 illustrates the interaction between a **classical computer** and the **quantum processor housed in a dilution refrigerator**. The classical computer generates **IQ-modulated microwave signals** (in-phase I and quadrature Q) using a **local oscillator**, low-pass filters (LPF), and mixers to control the qubits. These signals are transmitted through **attenuated and thermally managed microwave lines** inside the dilution refrigerator, ensuring minimal noise interference. The qubit signals are amplified using **Josephson Parametric Amplifiers (JPA)** at the millikelvin stage and further boosted by **High Electron Mobility Transistors (HEMT) at 3K** before reaching room temperature for digitization and processing by the classical computer. The setup includes different thermal stages $(300K, 60K, 3K, 1K, 10mK)$ with materials like **CuNi and NbTi cables**, optimized for minimal heat load and signal integrity. This architecture ensures precise **control and readout of superconducting qubits** while maintaining quantum coherence.

Chapter 3

Design Methodology and Fabrication Considerations of the Proposed 4-Qubit Superconducting Quantum Chip

In the previous chapter, we discussed the theoretical background of superconducting quantum computing, including structure and characteristics of a Josephson Junction, different qubit structures etc. This chapter presents the design, tuning and fabrication considerations of the proposed 4-qubit chips.

3.1 Workflow

The workflow of this research begins with creating two chip designs: both consisting 2 transmon pockets and 2 transmon crosses, but one with ring-type coupling and another with qubits coupled to a central bus resonator. The process involves tasks like tuning components using LOM and Eigenmode analysis, verifying modes, and calculating surface participation ratios. Finally, the two designs are compared for aspects like qubit connectivity and required swaps. This provides a structured approach to optimizing quantum chip performance. Overall workflow of this research is shown in the process chart of Figure 3.1.

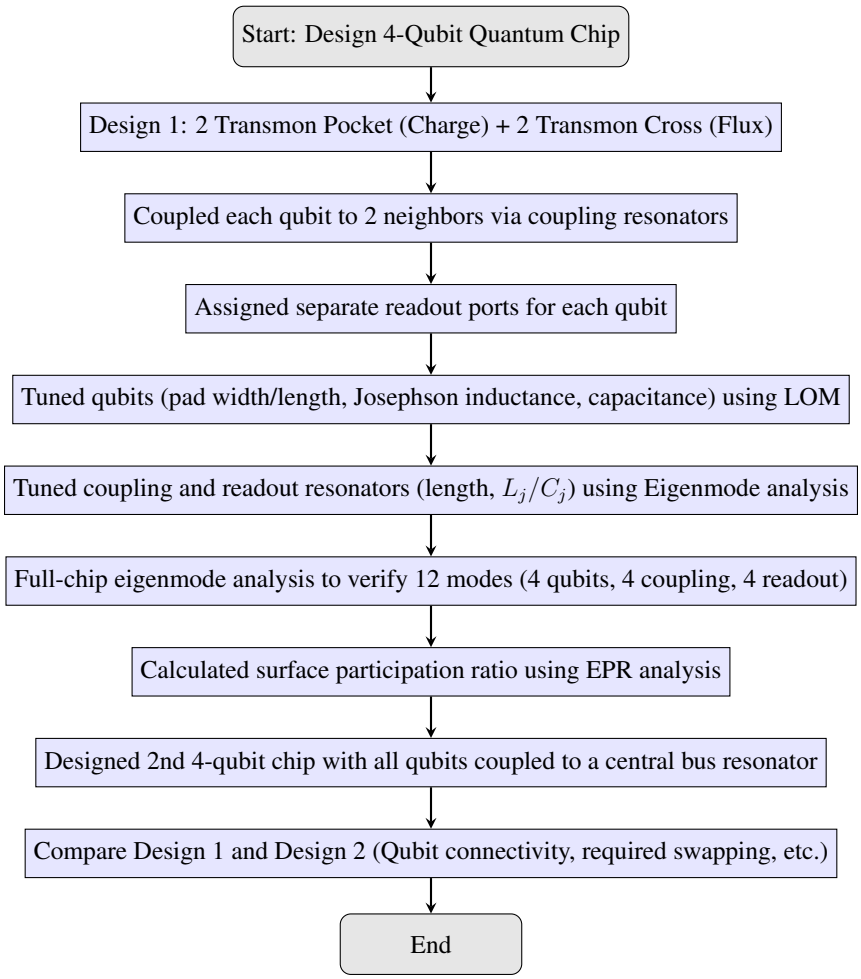


Figure 3.1: Process Chart of the overall workflow.

3.2 Circuit Design

For constructing a superconducting quantum circuit, **Qiskit Metal** is used. In Qiskit Metal, there are various types of qubits, including Transmon Pocket Qubit with or without charge/drive line, Transmon Cross Qubit with or without flux line, Concentric Transmon Qubit, Star Qubit etc. Some of these qubits are shown in Figure 3.2.

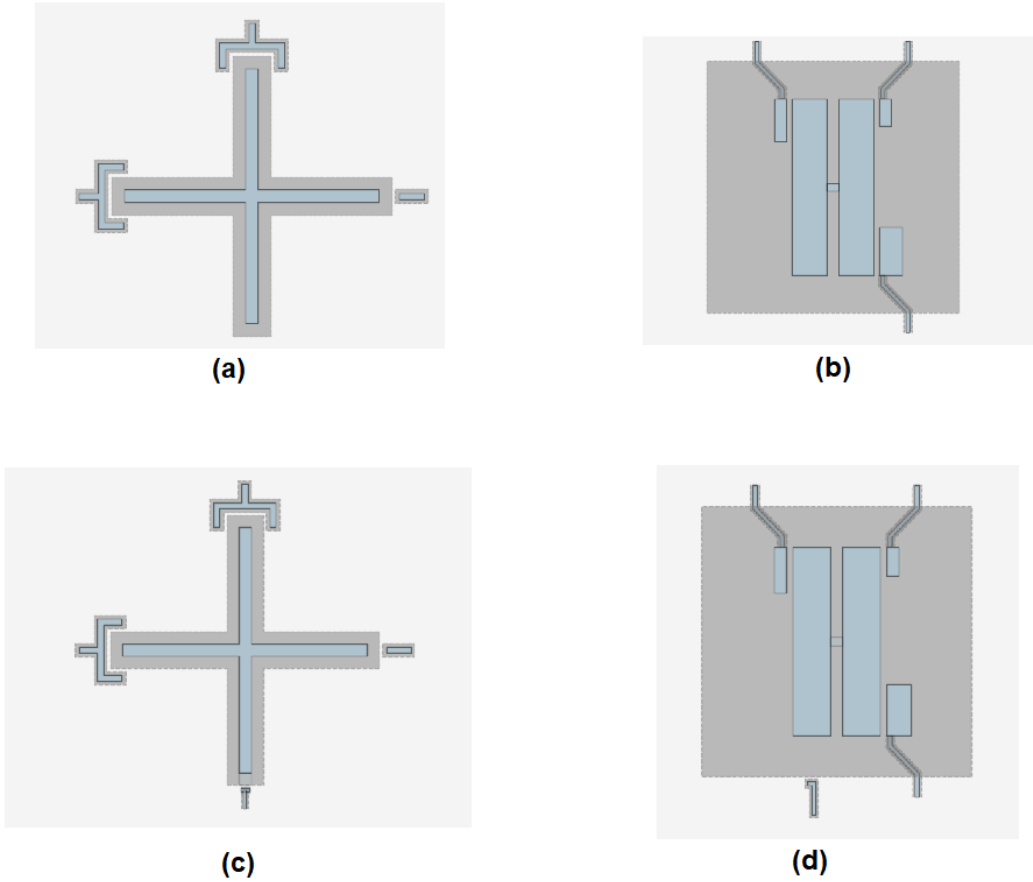


Figure 3.2: Different Types of Superconducting Qubit Structure - (a) Single Transmon Grounded or Xmon (**TransmonCross**), (b) Single Transmon - Floating (**TransmonPocket**), (c) Single Transmon Grounded (Xmon) Flux Lines (**TransmonCrossFL**), (d) Single Transmon - Floating with Charge Line (**TransmonPocketCL**)

Structural and characteristic difference among these are listed in table shown in Figure 3.3. Apart from these structures there are several other structures of superconducting qubits in the Metal library. Moreover, one can create a custom superconducting qubit structure using Qiskit Metal.

Feature	Transmon Pocket (Floating)	Transmon Pocket CL (Floating + Charge Line)	Transmon Cross (Grounded Xmon)	Transmon Cross FL (Grounded Xmon + Flux Line)
Structure	Floating capacitor-based design	Floating capacitor with charge line	Cross-shaped capacitor (Xmon)	Cross-shaped capacitor (Xmon) with flux bias
Grounding	Floating	Floating	Grounded	Grounded
Control Lines	No direct control line	Charge line added for direct frequency control	No charge line, but capacitive coupling	Flux bias line for tunability
Junction Type	Single Josephson Junction (JJ)	Single JJ with charge control	Single JJ (Xmon structure)	SQUID (two JJs) for flux tunability
Readout & Coupling	Capacitive coupling to resonator	Charge line allows external frequency tuning	Readout via capacitive coupling	Flux bias enables tunability of transition frequency
Frequency Tunability	Fixed after fabrication	Some tunability via charge line	Fixed due to grounding	Tunable via SQUID loop (magnetic field control)
Noise Susceptibility	Charge noise due to floating	Charge noise, but mitigated by charge line	Lower charge noise (grounded)	Susceptible to flux noise from external field
Decoherence Sources	Charge noise and dielectric losses	Charge noise + external drive effects	Lower charge noise, but dielectric loss	Flux noise + dielectric loss
Applications	Standard transmon, used in 3D architectures	Charge-controlled qubits for improved coherence	Xmon qubits for high coherence	Tunable qubits for quantum annealing, parametric control

Figure 3.3: Comparison between different Types of Transmon Structures

To couple two qubits there exists different types of couplers in the Metal library, for example -

- Direct Coupler
- Bus Resonator Coupler
- Tunable Coupler (designed in MIT)

3.2.1 Proposed Design-1

Qubit Geometry: The first design consists of two Transmon Pocket qubits with charge line and two Transmon Cross with flux line.

Coupling: Each qubit is coupled with 2 neighbouring qubits via bus coupling resonators.

Readout: Each qubit has seperate readout resonator. Readout port of the qubits are coupled to the readout launchpads via an **Interdigitated Capacitor**.

Charge Line and Flux Lines: The transmon pocket qubits have charge line and transmon cross qubits have flux lines. Each charge line and flux line is connected to seperate launchpads for convenience.

Figure 3.4 shows the first design of a 4-Qubit Superconducting Chip.

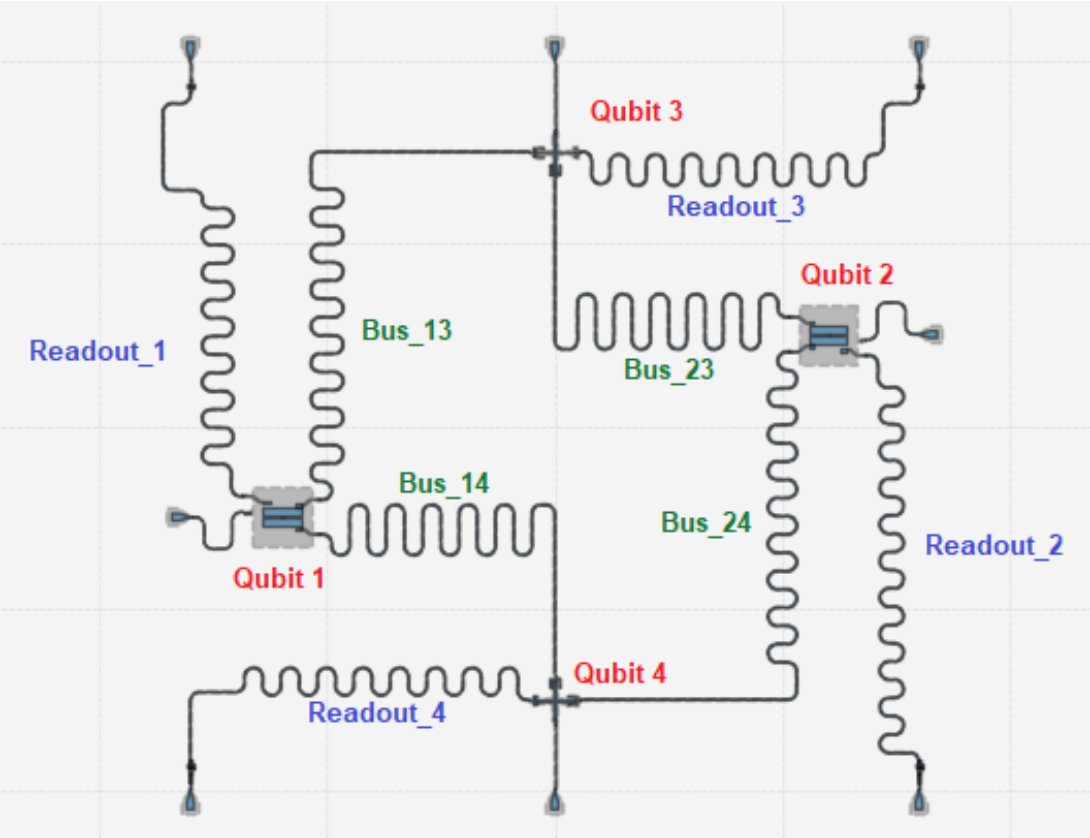


Figure 3.4: 4 Qubit Chip: Design 1

On the other hand, Figure 3.5 shows the same design, rendered in Ansys.

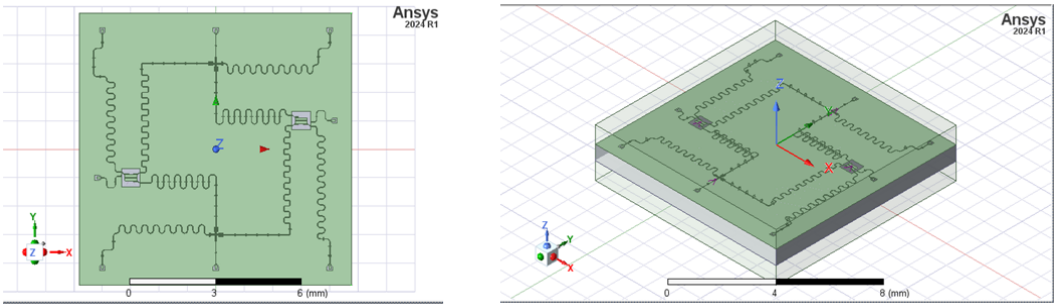


Figure 3.5: Rendered in Ansys: 4 Qubit Chip - Design 1

3.2.2 Proposed Design-2

Qubit Geometry: The second design consists of two Transmon Pocket qubits with charge line and two Transmon Cross with flux line, similar to the first design.

Coupling: All qubits are coupled to a central bus resonator via an **Interdigitated Tee Capacitor**. In this structure, qubits do not directly couple to each other but exchange information via the bus.

Readout: Each qubit has seperate readout resonator. Readout port of the qubits are coupled to the readout launchpads via an **Interdigitated Capacitor**, similar to the first design.

Charge Line and Flux Lines: The transmon pocket qubits have charge line and transmon cross qubits have flux lines. Each charge line and flux line is connected to seperate launchpads for convenience, similar to the first design. par Figure 3.6 shows the Design-2 of a 4-Qubit Superconducting Chip.

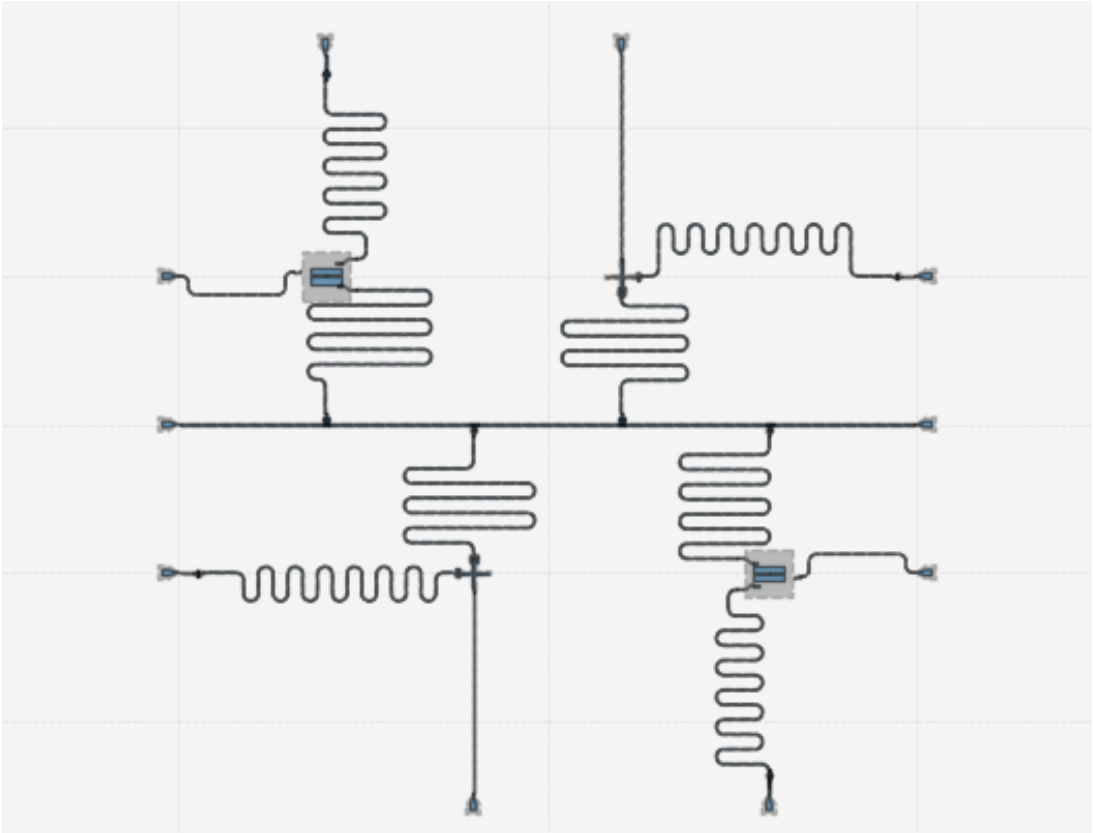


Figure 3.6: 4 Qubit Chip: Design 2

3.3 Circuit Tuning and Method of Analysis

The Hamiltonian of a superconducting quantum circuit system is fundamental in describing its dynamic evolution, making its accurate quantification essential for chip design. Various methods have been developed for this purpose, with the Lumped Oscillator Model (LOM) and Energy Participation Ratio (EPR) being among the most widely used approaches. Yuan et al. [28] validated the applicability of these methods to the design of 2D transmon quantum chips, showing that LOM provides greater data diversity and more precise qubit frequency calculations due to its comprehensive consideration of system couplings. Given that superconducting quantum chips function as open quantum systems, determining their Hamiltonian is crucial for deriving their dynamics. Several quantization techniques, including *impedance-based black-box quantization*, *LOM*, and *EPR*, have been proposed to refine these calculations, aiding in the optimization of superconducting quantum chip architectures [67–69].

3.3.1 Analysis Method: LOM

The lumped oscillator model (LOM) method is derived based on the quantization of the lumped model. In the LOM method, the distributed microwave circuit is equivalented to a lumped circuit [70], and the computational efficiency is higher than the full-wave method [68]. Its core idea is to divide the physical layout of a quantum processor into disjoint units, each of which can independently extract electrical parameters. Yuan et al. [28] described the theoretical framework and methodology for analyzing a superconducting quantum system consisting of qubits, readout resonators, and a transmission bus. The system's Hamiltonian is formulated by considering interactions between subsystems, incorporating effective capacitance and inductance values. These values are structured into a capacitance matrix, derived using Ansys Q3D Extractor, to compute qubit frequencies accurately. Figure 3.7 shows the layout of a qubit coupled to a readout resonator, the Q3D cell model, the capacitance matrix and the LOM analysis method.

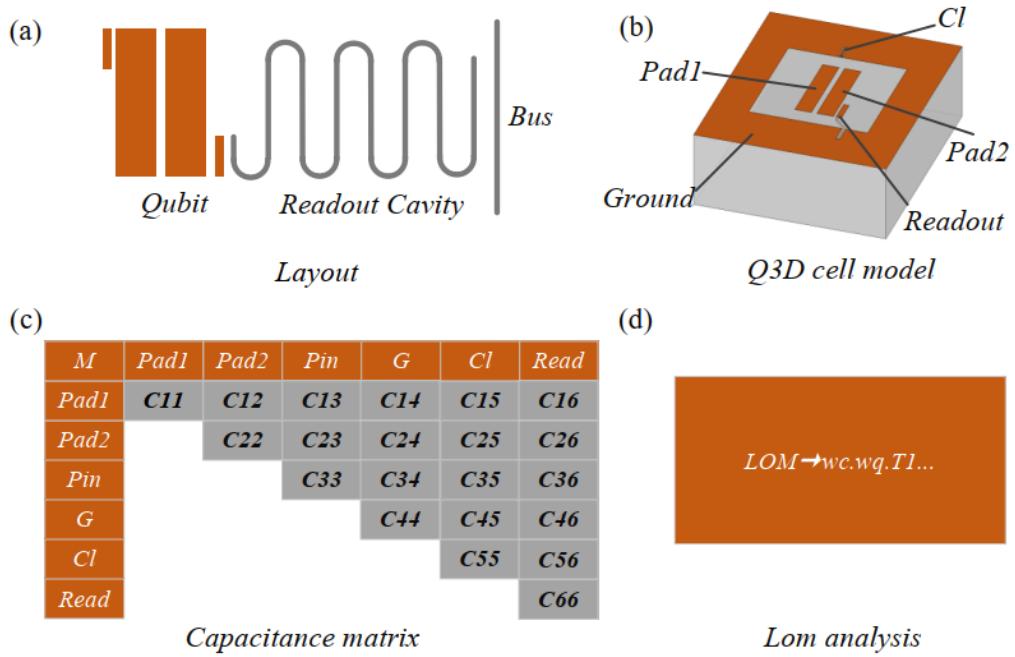


Figure 3.7: (a) the physical layout of a qubit coupled to a readout resonator, (b) the Q3D model of a transmon qubit, (c) the capacitance matrix, and (d) LOM analysis. The capacitance matrix plays a crucial role in quantifying qubit coupling and self-capacitance, ensuring precise calculations for quantum system design. [28]

In the LOM method, the qubit frequency is determined by the following formula:

$$\omega_q = \frac{1}{\sqrt{L_j C_q}} - E_c \tag{3.1}$$

where E_c is the capacitance energy of the superconducting qubit, L_j is the inductance of the Josephson junction, and C_q is the capacitance of the qubit. The relationship between E_c and C_q is

$$E_c = \frac{e^2}{2C_q} \tag{3.2}$$

The inductance of the Josephson junction can be determined by the magnetic flux quantum and the critical current,

$$L_j = \frac{\Phi_0}{I_c}. \tag{3.3}$$

In this method, equivalent capacitance and inductance and length-frequency relation for a full-wavelength, half wavelength and a quarter-wavelength readout resonator are given in the table 3.1, where ω_c is the frequency of the resonator, and Z_c represents the characteristic impedance of the resonator.

Table 3.1: Capacitance, Inductance, and Length-Frequency Relations for Different Resonators

Resonator Type	**Capacitance** C	**Inductance** L	**Length-Frequency Relation**
Quarter-Wave	$\frac{\pi}{4\omega_c Z_c}$	$\frac{1}{\omega_c^2 C'}$	$\omega_c = \frac{c\pi}{2l}\sqrt{\frac{2}{\varepsilon_r+1}}$
Half-Wave	$\frac{\pi}{2\omega_c Z_c}$	$\frac{1}{\omega_c^2 C'}$	$\omega_c = \frac{c\pi}{l}\sqrt{\frac{2}{\varepsilon_r+1}}$
Full-Wave	$\frac{\pi}{\omega_c Z_c}$	$\frac{1}{\omega_c^2 C'}$	$\omega_c = \frac{2c\pi}{l}\sqrt{\frac{2}{\varepsilon_r+1}}$

3.3.2 Analysis Method: EPR

The Energy Participation Ratio (EPR) method is a computational framework that quantifies how different circuit elements contribute to the system's total energy. It assigns a fraction of the stored electromagnetic energy to each component, enabling precise calculations of qubit frequencies, anharmonicity, coupling strengths, and dielectric losses, which are crucial for coherence. Derived from first principles, EPR transforms circuit quantization into an energy distribution problem, where the *energy participation ratio* (P_{mj}) represents the fraction of a mode's energy stored in a given circuit element. This ratio is fundamental in resolving the system's Hamiltonian and constructing the many-body quantum model. The Hamiltonian of a system of simple Transmon qubits coupled directly to a readout resonator $\hat{H}_{full}$ can be divided into linear and nonlinear parts:

$$\hat{H}_{full} = \hat{H}_{lin} + \hat{H}_{nl} \tag{3.4}$$

where $\hat{H}_{lin}$ is composed of the linear energy part of the Josephson junction and the linear energy correlation term of the resonator, and $\hat{H}_{nl}$ is the nonlinear energy correlation term of the Josephson junction. The **linear Hamiltonian** ($\hat{H}_{lin}$) represents the harmonic energy of the system, describing the independent contributions of the resonator and qubit modes using their respective annihilation operators. It includes terms proportional to their angular frequencies, ω_c and ω_q, which characterize the eigenmode frequencies of the resonator and qubit, respectively. On the other hand, the **nonlinear Hamiltonian** ($\hat{H}_{nl}$) accounts for the Josephson junction's nonlinearity, incorporating a cosine potential term that leads to anharmonicity in the qubit spectrum. This nonlinearity arises due to quantum zero-point fluctuations in both the cavity and qubit modes, denoted by φ_c and φ_q, which can be determined using the **Energy Participation Ratio (EPR)** method. The nonlinear component is essential for defining the transmon qubit's anharmonicity, enabling controlled quantum operations.

The **effective Hamiltonian** of the system is obtained by approximation:

$$\hat{H}_{eff} = (\omega_q - \Delta_q)\hat{n}_q + (\omega_c - \Delta_c)\hat{n}_c - \chi_{qc}\hat{n}_q\hat{n}_c - \frac{1}{2}\alpha_q\hat{n}_q(\hat{n}_q - \hat{\mathbb{I}}) - \frac{1}{2}\alpha_c\hat{n}_c(\hat{n}_c - \hat{\mathbb{I}}) \quad (3.5)$$

where $\hat{n}_q = \hat{a}_q^\dagger\hat{a}_q$ and $\hat{n}_c = \hat{a}_c^\dagger\hat{a}_c$ represent the cavity and qubit particle number operators, respectively, Δ_q is the **Lamb shift** of the qubit frequency, α_q is the **anharmonicity** of the qubit, and χ_{qc} is the **dispersion shift**. The qubit's anharmonicity (α_q) and the cavity's self-Kerr effect (α_c) are derived from the energy participation ratio (EPR), quantifying how much of the system's energy is stored in each component.

$$\alpha_q = \frac{1}{2}\chi_{qq} = p_q^2\frac{\hbar\omega_q^2}{8E_j} \quad (3.6)$$

$$\alpha_c = \frac{1}{2}\chi_{cc} = p_c^2\frac{\hbar\omega_c^2}{8E_j} \quad (3.7)$$

Additionally, the dispersive coupling term (χ_{qc}) represents the frequency shift caused by the interaction between the qubit and the resonator.

$$\chi_{qc} = p_q p_c\frac{\hbar\omega_q\omega_c}{4E_j} \quad (3.8)$$

Finally, the sum of the energy participations must satisfy $p_q + p_c = 1$, ensuring proper energy distribution in the system.

3.3.3 Comparison of Lumped Oscillator Model and Energy Participation Ratio Methods in Designing Two-Dimensional Superconducting Quantum Chips

A comparative analysis between both the methods are shown in Table 3.2.

Table 3.2: Comparison between EPR Method and Lumped Oscillator Model

Feature	**EPR Method**	**Lumped Oscillator Model**
Energy distribution approach	Uses field simulations	Uses circuit parameters
Qubit frequency accuracy	High	Moderate
Resonator frequency accuracy	High	Moderate
Loss analysis	Explicitly considers dielectric losses	Less direct loss estimation
Computation	Requires electromagnetic simulations	Analytical calculations
Timing consideration	Computationally expensive	Faster analytical approach

3.4 Fabrication Considerations

After the tuning is complete, a **GDS** file needs to be prepared to create a mask and fabricate a chip. Firstly, a gds render instance has to be created. When preparing a superconducting qubit chip for fabrication using Qiskit Metal's GDS renderer, several key considerations must be addressed to ensure compatibility with lithography processes. The bounding box scaling factors can expand or shrink the design area, preventing clipping issues, while short segment adjustments help refine small feature transitions and avoid unnecessary fillets in CPW bends. Josephson junction placement is controlled by **junction_pad_overlap** which ensures proper connectivity between the junction and superconducting pads, a critical factor in maintaining circuit performance. The cheese pattern is used to mitigate stress in the chip during fabrication, preventing unwanted defects. The ground plane setting is enabled to suppress spurious microwave modes, essential for high-coherence qubit operation. Additionally, the GDS file export precision ensures nanometer-scale accuracy, while **width_LineString** defines the minimum metal linewidth, which must align with the foundry's process constraints. The circuit's geometry is refined using *corners* = *circular bend* to minimize RF reflections, improving transmission line performance. Before finalizing the GDS export, it is crucial to validate the design in a GDS viewer like **KLayout** to check layer assignments, feature sizes, and cheese hole placements, ensuring the chip is fabrication-ready.

An example GDS view of proposed design 1 in KLayout is shown in Figure 3.8.

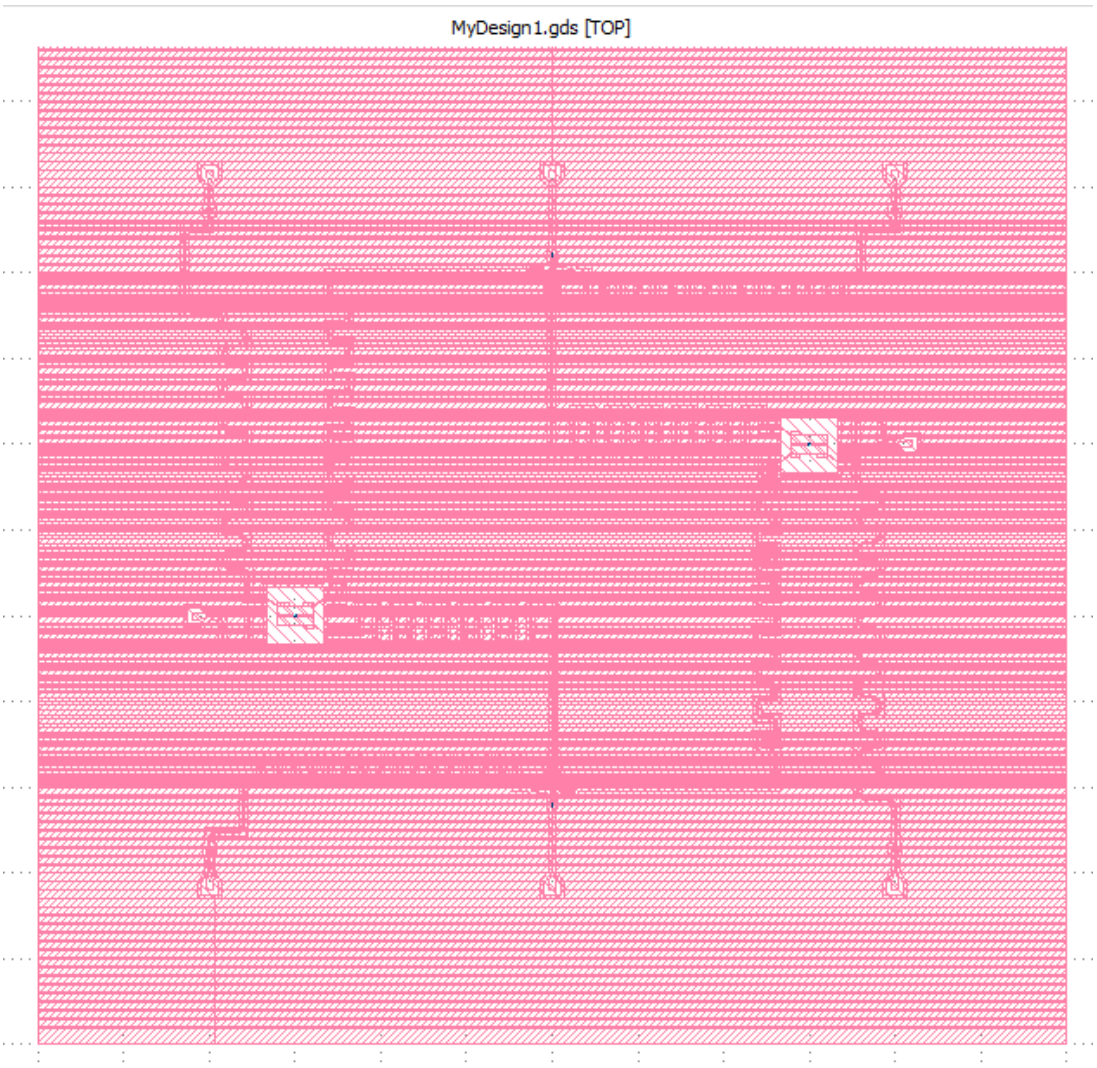


Figure 3.8: GDS rendered view of the Proposed Design-1

Chapter 4

Results and Discussion

The previous chapter described the design procedure of two-Qubit Superconducting Quantum Chips. This chapter describes the performance and tuning of one of the designed chips using various simulation analysis.

4.1 Tuning The Qubits using LOM Analysis

The desired range of qubit properties is extracted and listed in the Table 4.1. The literature are also included in this table.

Table 4.1: Key Transmon Qubit Parameters with References

Parameter	**Typical Range**	**Importance**	**Reference**
Qubit Frequency (ω_q)	4 GHz – 7 GHz	Ensures efficient control and readout	Koch et al., 2007 [20]
Anharmonicity (α_q)	100 MHz – 400 MHz	Prevents leakage to higher levels	Barends et al., 2013 [22]
Relaxation Time (T_1)	10 µs – 500 µs	Measures qubit lifetime	Place et al., 2021 [71]
Dispersive Shift (χ_{qc})	-1 MHz to -5 MHz	Enables qubit readout	Walter et al., 2017 [72]
Qubit-Bus Coupling (g_{q-bus})	10 – 80 MHz	Controls qubit interaction with resonator	Blais et al., 2020 [58]

Therefore, for the purpose of the design, the target values are listed in the table 4.2.

The dispersive shift (χ) is primarily important for the readout resonator, as it enables **quantum non-demolition (QND) readout** by shifting the resonator frequency depending on the qubit state, allowing efficient state measurement.

Table 4.2: Target Values for Qubit Parameters

Parameter	**Target Value**
Frequency (ω_q)	Qubit 1 = 4.8 GHz Qubit 2 = 5 GHz Qubit 3 = 5.2 GHz Qubit 4 = 5.4 GHz
Anharmonicity (α_q)	∼300 MHz
Dispersive Shift (χ_{qc})	∼1 MHz
Qubit-bus Coupling (g_{q-bus})	∼80 MHz

4.1.1 Tuning Qubit 1

Before tuning the qubit, lets look at some important default parameters of Transmon Pocket Qubit with Charge Line are listed in the table 4.3.

Table 4.3: Qubit 1 Default Parameters including Connection Pads, Bus Connections, and other Qubit Properties

Category	**Parameter**	**Value**
Connection Pads		
	readout_pad_gap	50um
	pad_width	80um
	pad_height	30um
	pad_cpw_shift	5um
	pad_cpw_extent	25um
	cpw_extend	100um
	pocket_extent	5um
	pocket_rise	65um
Bus Resonator		
	pad_gap	10um
	pad_width	60um
	pad_height	30um
	pad_cpw_shift	5um
	pad_cpw_extent	25um
	cpw_extend	100um
	pocket_extent	5um
	pocket_rise	65um
Other Qubit Parameters		
	pad_gap	30um
	inductor_width	20um
	pad_width	425um
	pad_height	90um
	pocket_width	650um
	pocket_height	650um
	make_CL	True
	cl_gap	60um
	cl_width	10um
	cl_length	120um
	cl_ground_gap	6um
	cl_pocket_edge	0
	cl_off_center	50um

The tuning of the qubits were performed by modifying the geometric configuration of the qubits or by changing Josephson Junction properties. The results of the tuning iterations are listed in the table shown in Figure 4.1.

Serial No.	Modifications	Parameter	Target Value	Resulted Value
1	**Geometric Modification:** (Default) • Qubit Pad Gap = 30um, • Readout Pad Gap = 50um, • Readout Pad Width = 80um, • Bus_13 Pad Width = 60um, • Bus_14 Pad Width = 60um Lj = 14.9 nH, Cj= 2fF,	Frequency	4.8 GHz	4.917 GHz
		Anharmonicity	~300 MHz	-375.09 MHz
		Dispersive Shift	~1 MHz	-0.23 MHz
		Qubit-bus Coupling	~80 MHz	**Readout:** -35.14 MHz
				Bus_13: -49.17 MHz
				Bus_14: 53.65 MHz
2	**Geometric Modification:** • Qubit Pad Gap = 28um, • Readout Pad Gap = 20um, • Readout Pad Width = 100um, • Bus_13 Pad Width = 130um, • Bus_14 Pad Width = 130um Lj = 14.9 nH, Cj= 2fF	Frequency	4.8 GHz	4.794 GHz
		Anharmonicity	~300 MHz	-352.829 MHz
		Dispersive Shift	~1 MHz	-0.544 MHz
		Qubit-bus Coupling	~80 MHz	**Readout:** -59.327 MHz
				Bus_13: -80.695 MHz
				Bus_14: 93.068 MHz
3	**Geometric Modification:** • Qubit Pad Gap = 28um, • Readout Pad Gap = 15um, • Readout Pad Width = 150um, • Bus_13 Pad Width = 130um, • Bus_14 Pad Width = 110um Lj = 14.9 nH, Cj= 2fF	Frequency	4.8 GHz	4.783 GHz
		Anharmonicity	~300 MHz	-350.86 MHz
		Dispersive Shift	~1 MHz	-1.127 MHz
		Qubit-bus Coupling	~80 MHz	**Readout:** -86.008 MHz
				Bus_13: -75.86 MHz
				Bus_14: 84.94 MHz
4	**Geometric Modification:** • Qubit Pad Gap = 28um, • Readout Pad Gap = 15um, • Readout Pad Width = 150um, • Bus_13 Pad Width = 130um, • Bus_14 Pad Width = 110um Lj = 16 nH, Cj= 2fF	Frequency	4.8 GHz	4.604 GHz
		Anharmonicity	~300 MHz	-353.826 MHz
		Dispersive Shift	~1 MHz	-0.942 MHz
		Qubit-bus Coupling	~80 MHz	**Readout:** -84.49 MHz
				Bus_13: -74.519 MHz
				Bus_14: 83.439 MHz
5	**Geometric Modification:** • Qubit Pad Gap = 28um, • Readout Pad Gap = 15um, • Readout Pad Width = 150um, • Bus_13 Pad Width = 130um, • Bus_14 Pad Width = 110um Lj = 14.9 nH, Cj= 4 fF	Frequency	4.8 GHz	4.717 GHz
		Anharmonicity	~300 MHz	-339.285 MHz
		Dispersive Shift	~1 MHz	-1.05 MHz
		Qubit-bus Coupling	~80 MHz	**Readout:** -86.66 MHz
				Bus_13: -76.43 MHz
				Bus_14: 85.58 MHz

Figure 4.1: Tuning for Qubit 1.

From the table shown in Figure 4.1, it can be stated that results shown in the **third row** is the closest to our target tuning parameter values. Moreover, effects of the changes in geometry or

inductance or capacitance is shown in figure 4.2.

Modified Parameter	Change	Effect on Frequency	Effect on Anharmonicity	Effect on Qubit-Bus Coupling	Effect on Dispersive Shift
L_j ↑	14.9nH → 16nH	↓	↓	↓	↓
C_j ↑	2fF → 4fF	↑	↑	↑	↓
Qubit Pad Gap ↓	30µm → 28µm	↓	↓	↓	↓
Bus_13 Pad Width ↑	60µm → 130µm	↓	↓	↓	↓
Bus_14 Pad Width ↑	60µm → 130µm	↓	↓	↓	↓
Bus_14 Pad Width ↓	130µm → 110µm	↓	↓	↓	↓
Readout Pad Width ↑	80µm → 150µm	↓	↓	↓	↓
Readout Pad Gap ↓	50µm → 15µm	↓	↓	↓	↓

Figure 4.2: Effect of Changes in Tuning for Qubit 1.

The extracted **capacitance matrix** for the tuned qubit (according to the values of the *third row* of Table shown in Figure 4.1) is shown in Table 4.4.

Table 4.4: Capacitance Matrix for Superconducting Qubit Circuit

$$\mathbf{C} = \begin{bmatrix} & \text{bus\_13} & \text{bus\_14} & \text{cl\_metal} & \text{ground} & \text{pad\_bot} & \text{pad\_top} & \text{readout} \\ \text{bus\_13} & 53.33077 & -0.33159 & -0.00965 & -34.13588 & -1.63182 & -15.89420 & -0.48710 \\ \text{bus\_14} & -0.33159 & 50.24319 & -0.00780 & -33.39629 & -14.15842 & -1.42492 & -0.14047 \\ \text{cl\_metal} & -0.00965 & -0.00780 & 16.52001 & -15.93983 & -0.13809 & -0.18193 & -0.16703 \\ \text{ground} & -34.13588 & -33.39629 & -15.93983 & 249.39844 & -38.99769 & -33.19551 & -35.40139 \\ \text{pad\_bot} & -1.63182 & -14.15842 & -0.13809 & -38.99769 & 92.28256 & -32.74671 & -1.84476 \\ \text{pad\_top} & -15.89420 & -1.42492 & -0.18193 & -33.19551 & -32.74671 & 101.60921 & -15.69293 \\ \text{readout} & -0.48710 & -0.14047 & -0.16703 & -35.40139 & -1.84476 & -15.69293 & 54.65882 \end{bmatrix}$$

The convergence plots of frequency and anharmonicity of Qubit-1 are shown in the Figure 4.3.

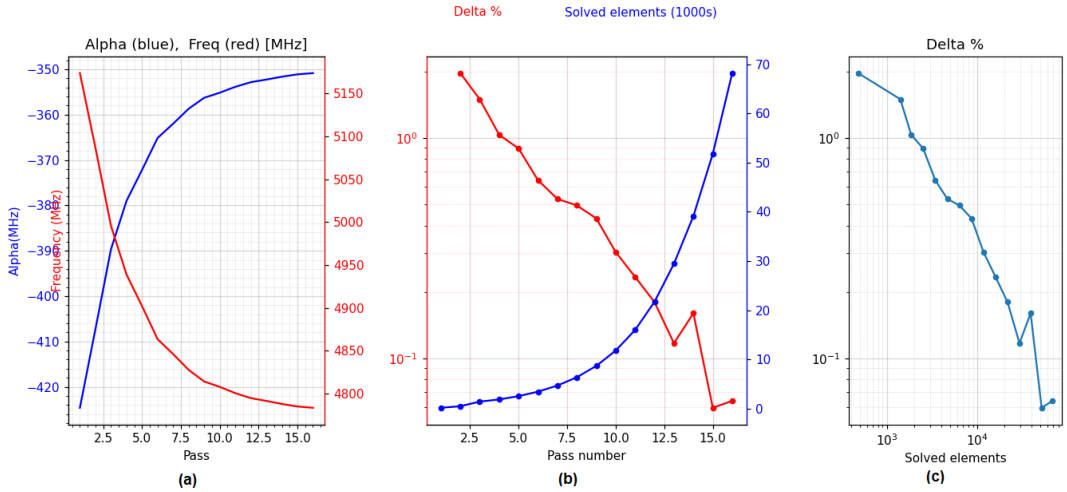


Figure 4.3: Convergence Plots for Qubit 1.(a)Convergence of Frequency and Anharmonicity with Pass Number, (b)Change in Delta and Solved Elements vs. Pass Number, (c)Delta Percentage vs. Number of Solved Elements

The first graph in figure 4.3, **"Convergence of Frequency and Anharmonicity with Pass Number"** illustrates how the qubit **frequency (red)** and **anharmonicity (blue)** stabilizes as the simulation progresses through multiple passes. The second graph, **"Change in Delta and Solved Elements vs. Pass Number,"** shows how the **relative error** ($\Delta\%$) (red) decreases while the **number of solved elements** (blue) rises exponentially, indicating simulation refinement. The third graph, **"Delta Percentage vs. Number of Solved Elements"** highlights how $\Delta\%$ decreases with increasing solved elements, demonstrating the diminishing rate of improvement beyond a certain threshold. From these graphs collectively, it can be stated that the parameters-frequency and anharmonicity are **well converged after 15 passes**. Thus, these graphs help estimate the **optimal pass number** for balancing computational efficiency and accuracy.

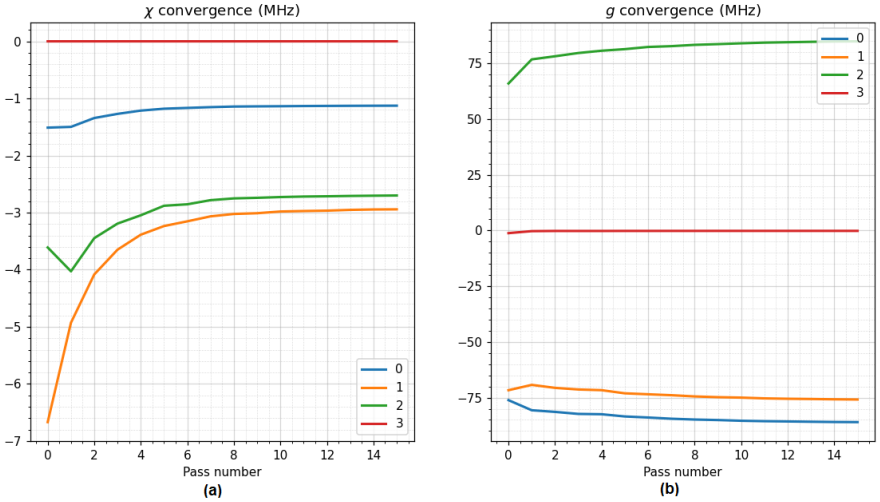


Figure 4.4: Convergence Plots for Qubit 1.(a)Convergence of χ, (b)Covergence of g

In figure 4.4, the left graph, **"χ Convergence (MHz)"** shows the dispersive shift (χ) of four different modes as a function of pass number, where most modes stabilize after a few passes, indicating convergence in qubit-resonator interaction strength. Of all the modes, we can distinguish the readout resonator mode which is the **blue curve** converging near -1. The right graph, **"g Convergence (MHz)"** represents the coupling strength (g) across different modes, showing that while some modes maintain near-zero values, others stabilize at positive or negative values, highlighting distinct coupling behaviors.

4.1.2 Tuning Qubit 2, Qubit 3 and Qubit 4

Similar to the process shown in Sub-section 4.1.1, rest of the Qubits are tuned and their capacitance matrix can be extracted along with similar convergence plots. Figure 4.5 shows the updated paramaters of the Qubits after tuning.

Qubit	Modifications	Parameter	Target Value	Resulted Value
Qubit 2	**Geometric Modification:** • Qubit Pad Gap = 28um, • Readout Pad Gap = 15um, • Readout Pad Width = 130um, • Bus_13 Pad Width = 100um, • Bus_14 Pad Width = 110um Lj = 13.3 nH, Cj= 2 fF	Frequency	5 GHz	5.032 GHz
		Anharmonicity	~300 MHz	-338.49 MHz
		Dispersive Shift	~1 MHz	-1.085 MHz
		Qubit-bus Coupling	~80 MHz	**Readout:** -83.97 MHz
				Bus_23: 83.99 MHz
				Bus_24: -77.89 MHz
Qubit 3	**Geometric Modification:** • Qubit Cross Width = 18um, • Bus_13 Claw Width = 20um, • Bus_13 Claw Length = 120um, • Bus_14 Claw Width = 20um, • Bus_14 Claw Length = 120um, Lj = 8 nH, Cj= 3 fF	Frequency	5. GHz	5.22 GHz
		Anharmonicity	~300 MHz	-288.49 MHz
		Dispersive Shift	~1 MHz	-0.985 MHz
		Qubit-bus Coupling	~80 MHz	**Readout:** -82.27 MHz
				Bus_23: 79.59 MHz
				Bus_24: -81.45 MHz
Qubit 4	**Geometric Modification:** • Qubit Cross Width = 18um, • Bus_13 Claw Width = 20um, • Bus_13 Claw Length = 120um, • Bus_14 Claw Width = 20um, • Bus_14 Claw Length = 120um, Lj = 7.5 nH, Cj= 3 fF	Frequency	5.4 GHz	5.474 GHz
		Anharmonicity	~300 MHz	-345.56 MHz
		Dispersive Shift	~1 MHz	-1.118 MHz
		Qubit-bus Coupling	~80 MHz	**Readout:** -82.29 MHz
				Bus_23: 83.67 MHz
				Bus_24: -76.93 MHz

Figure 4.5: Table of Upadated Parameters after Tuning of Qubit 2, Qubit 3 and Qubit 4

4.2 Tuning Coupling and Readout Resonators using EPR

While designing the chip, the resonator length was determined using a classical formula. At first, the guided wavelength of a planar CPW transmission line is given by:

$$\lambda_g = \frac{\lambda_0}{\sqrt{\epsilon_{\text{eff}}}} = \frac{c}{f\sqrt{\epsilon_{\text{eff}}}} \tag{4.1}$$

where: λ_g is the guided wavelength, $\lambda_0 = \frac{c}{f}$ is the free-space wavelength, c is the speed of light, f is the operating frequency, ϵ_{eff} is the effective dielectric constant.

The resonator length is then determined as:

$$L = \frac{\lambda_g}{N} \tag{4.2}$$

where: L is the resonator length, N determines the fraction of λ_g (e.g., $N = 2$ for $\lambda/2$, $N = 4$ for $\lambda/4$).

4.2.1 Tuning the Bus Resonators

The table in Figure 4.6 shows the updated lengths of the coupling buses after tuning. Resonator tuning was performed using EPR analysis method.

Bus	Target Frequency (GHz)	Theoretical Value (mm)	Value after Tuning (mm)	Resulted Frequency (GHz)
Bus_13	5.8	10.65	9.5	5.808
Bus_23	6	10.297	9.2	5.961
Bus_24	6.2	9.965	9	6.13
Bus_14	6.4	9.653	8.5	6.417

Figure 4.6: Table of Upadated Lengths after Tuning of Coupler Buses

4.2.2 Tuning the Readout Resonators

The table in Figure 4.7 shows the updated lengths of the readout resonator buses after tuning. This process of tuning was also performed using EPR analysis method, similar to the coupling buses.

Readout	Target Frequency (GHz)	Theoretical Value (mm)	Value after Tuning (mm)	Resulted Frequency (GHz)
Readout_1	6.8	8.95	8.3	6.808
Readout _2	7	8.69	8	7.022
Readout _3	7.2	8.45	7.9	7.232
Readout _4	7.4	8.224	7.7	7.424

Figure 4.7: Table of Upadated Lengths after Tuning of Readout Resonators

While rendering the readout resonators, the interdigitated capacitor is neglected. But in a fabricated chip, the interdigitated capacitor can also affect the frequency of the resonator by a small amount.

4.3 Verification of the Tuning

For further verification of the tuning, the whole chip was rendered at once, which is computationally expensive. Then using eigenmode and EPR analysis all the mode frequencies were figured out using upto 10 passes. The resulting modes are listed in the table 4.5.

Table 4.5: Mode Frequencies in GHz

Mode	Freq. (GHz)
0	4.872064
1	5.073668
2	5.228010
3	5.242823
4	5.318098
5	5.455883
6	5.751513
7	5.953469
8	6.061440
9	6.380419
10	6.724154
11	6.829460

From the mode frequencies we can easily differentiate the qubits, bus resonators and readout resonators. Another important factor is the difference in mode frequencies are somewhat arount 200KHz (apart from mode 2 and mode 3). As the simulation is computationally quite expensive, the convergence quality is poor as shown in figure 4.8.

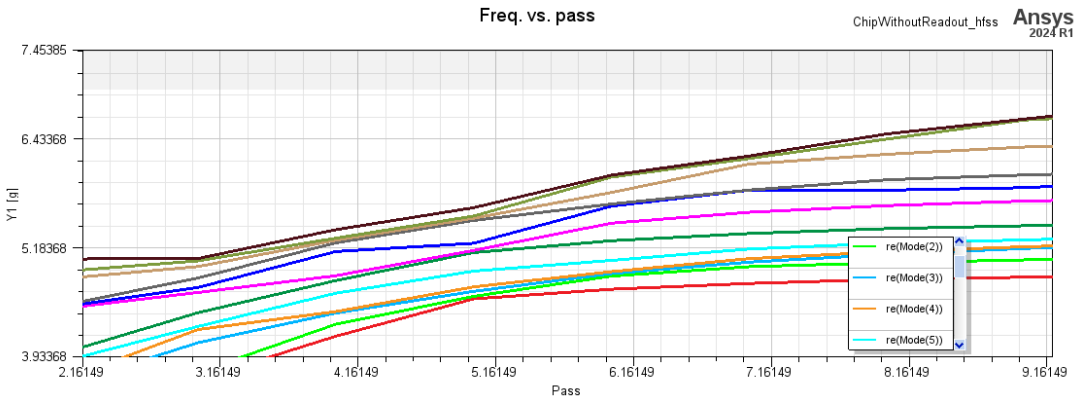


Figure 4.8: Convergence of all modes in Design 1

4.3.1 Calculation of Energy Participation ratio

The **Energy Participation Ratio (EPR) of the substrate** is defined as:

$$\text{EPR}_{\text{substrate}} = \frac{U_{\text{elec, substrate}}}{U_{\text{elec, total}}} \times 100\%$$

where $U_{\text{elec, substrate}}$ is the electric energy stored in the substrate, and $U_{\text{elec, total}}$ is the total electric energy in the system. In this case, the **EPR of the substrate is 91.8%** as shown in table 4.6, indicating that a significant portion of the electric energy resides in the substrate. A high substrate EPR suggests that dielectric losses in the substrate could limit qubit coherence, making it an essential factor in superconducting quantum circuit design.

Table 4.6: Energy Participation Ratio (EPR) Analysis of the Whole chip: Design 1

Parameter	**Value**
Total Electric Energy (energy_elec_all)	$5.58443313187822 \times 10^{-24}$ J
Substrate Electric Energy (energy_elec_substrate)	$5.12845080623539 \times 10^{-24}$ J
EPR of Substrate	91.8%
Magnetic Energy (energy_mag)	$2.33590433444508 \times 10^{-25}$ J
Magnetic Energy as % of Total Electric Energy	4.2%

4.4 Computational Cost

In this section, the computational cost for tuning a Qubit and a resonator is shown for various number of passes. Additionally, the overall computational cost for redering the tuning the whole chip at once is also mentioned.

4.4.1 Computational Cost for Tuning a Qubit

Convergence graphs for qubit 1 shown in figure 4.3 and 4.4 had maximum 15 passes with minimum error threshold as 0.05%. The same convergence plots were created varying number of maximum passes and minimum error threshold to figure out optimal number of passes and simulation time for each case was measured.

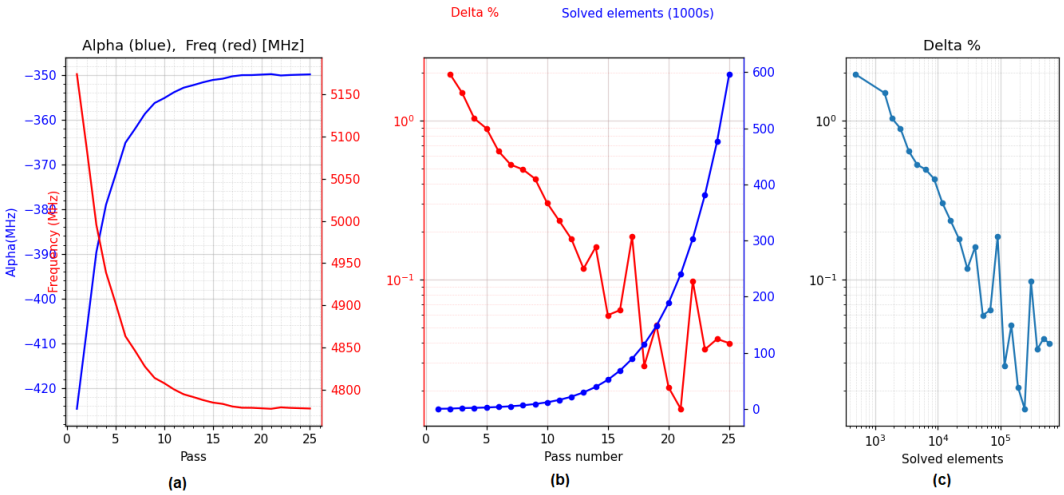


Figure 4.9: Convergence Plots for Qubit 1 for max passes = 25, min error threshold = 0.01%.(a)Convergence of Frequency and Anharmonicity with Pass Number, (b)Change in Delta and Solved Elements vs. Pass Number, (c)Delta Percentage vs. Number of Solved Elements

From the second graph in figure 4.9, it can be stated that, the values of frequency and anharmonicity of Qubit 1 have converged far better than the previous case. But the rate of change slows significantly after pass 15, showing diminishing improvement. In the previous case, the error percentage ($\Delta\%$) stabilized around pass 12-14, with fluctuations beyond this point. In the current case, $\Delta\%$ continues to decrease but exhibits high fluctuations after pass 18, suggesting over-refinement. Comparing the third graph in figure 4.9 with the previous case, it is visualized that both cases show a rapid drop in $\Delta\%$ initially, but after 10^4 solved elements, the decrease slows down.

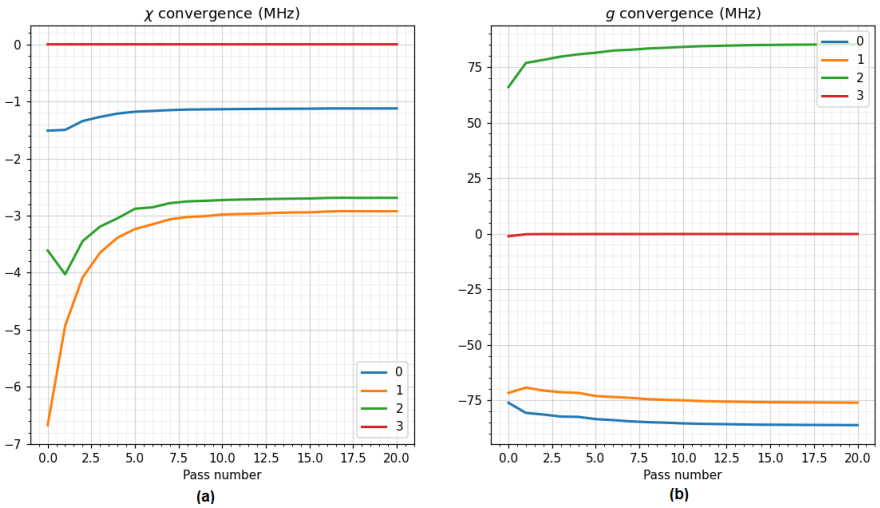


Figure 4.10: Convergence Plots for Qubit 1 for max passes =25, min error threshold = 0.01%.(a)Convergence of χ, (b)Covergence of g

Time taken for the LOM simulations of Qubit 1 are listed in the following table along with its convergence quality for various number of passes. Hardware used for the purpose of simulation in Ansys -

- **Processor:** AMD Ryzen 5 3600 6-Core Processor (3.60 GHz)
- **GPU:** NVIDIA GeForce GTX 1660 SUPER

Table 4.7: Convergence Behavior with Increasing max_pass Number for Qubit 1

Max no of passes	Time (seconds)	Convergence
10	42.8335	Poor
15	146.59	Moderate
21	620.338	Good
25	1521.411	Best

The data above can be plotted and fitted in an exponential model as shown in figure 4.11. So, it can be stated that with the increase in number of passes, the computational cost increases exponentially. The main metric for calculating computational cost is time required for simulation in our case.

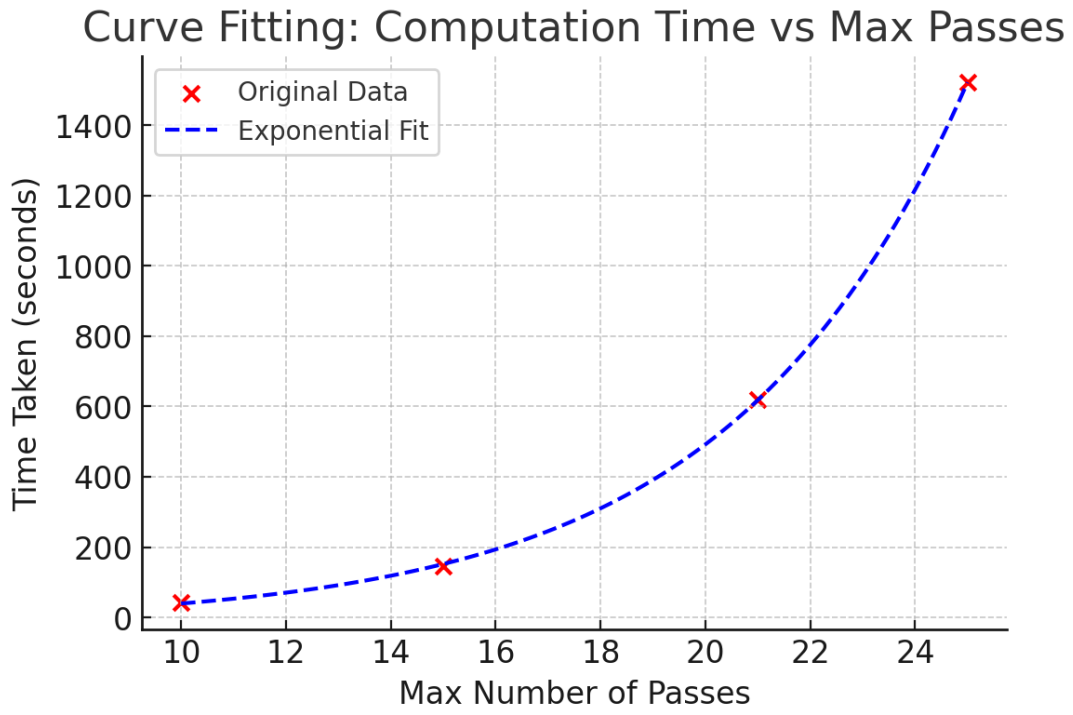


Figure 4.11: Time taken for LOM Simulation in seconds vs Number of Maximum passes for a Qubit

4.4.2 Computational Cost for Tuning a Resonator

For tuning a resonator using EPR analysis, time elapsed for simulation was measured **160.225 seconds for 12 passes**. The resulting convergence was satisfactory as shown in figure 4.12.

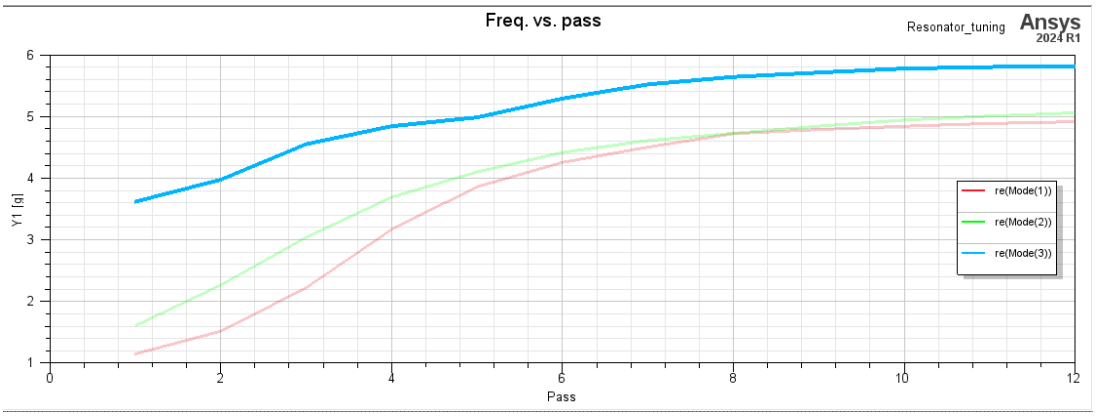


Figure 4.12: Convergence Plot for Resonator for max passes = 12

4.5 Comparison between Design 1 and Design 2

This section discusses a comparative analysis of Qubit-Connectivity between Design-1 and Design-2. In Design 1 each qubit is coupled to two neighbouring qubits, creating a Ring type architecture. The connectiviy diagram of Design 1 is shown in Figure 4.13.

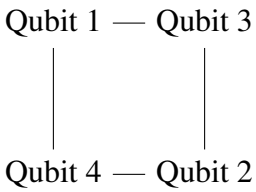


Figure 4.13: Connectivity Diagram of Design-1

Whereas the qubit connectiviy diagram of Design 2 is shown in Figure 4.14.

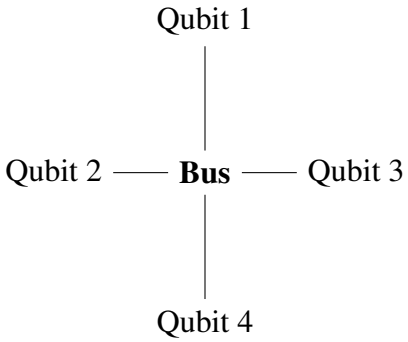


Figure 4.14: Connectivity Diagram of Design-2

As stated in Table 4.8, if 2 non neighbouring qubits in Design 1 has to interact, SWAP operations are needed, which increases error probability. In Design 2 on the other hand, SWAP operations are not required for any 2 qubits to interact. So Design 2, in this aspect, provides optimized coupling properties. A brief comparison regarding some other important aspects between the two designs have also been shown in this table .

Table 4.8: Comparison of Qubit Connectivity and Applications

Feature	**Design 1: Direct Coupling**	**Design 2: Bus Resonator Coupling**
Qubit Connectivity	Each qubit connects to two neighbors (e.g., a ring/chain).	Qubits interact through a shared **bus resonator**.
Gate Operations	**Two-qubit gates** (CZ, iSWAP, etc.) are executed via direct coupling.	Gates require **resonator-mediated** interactions.
Gate Feasibility (Q1-Q3)	No **direct interaction** $\rightarrow$ requires **SWAP gates** to move information through Q2 or Q4.	Q1 and Q3 can interact **directly through the bus**.
Crosstalk	Higher risk of **parasitic coupling** between qubits.	More **controlled coupling**, but can introduce **resonator-induced loss**.
Scalability	Extending to **more qubits** requires careful routing.	Easier to scale by adding qubits to the **bus**.
Error Accumulation	More SWAP gates = **higher error rates**.	No extra SWAP gates = **lower error**.
Readout Performance	Each qubit has a **dedicated readout**, avoiding interference.	Same **dedicated readout**, but bus interactions may introduce **readout crosstalk**.
Application Suitability	- Efficient for **nearest-neighbor algorithms** (e.g., VQE, QAOA). - Good for **local entanglement** operations.	- Better for **multi-qubit entanglement** (e.g., GHZ state preparation, quantum memories). - Useful for **global operations like quantum simulation**.
Fabrication Complexity	More inter-qubit wiring; requires precise **capacitance tuning**.	More straightforward fabrication, but **resonator design is crucial**.

Chapter 5

Conclusion and Scope for Future Work

In the previous chapters, a brief overview of design and tuning process of a small multi-qubit superconducting quantum chip was discussed along with two cusomized designs were proposed. In this chapter, concluding remarks of this work including some applications of a small scale multi-qubit chip will be discussed, followed by novelty of the work and a discussion on future aspects of this research.

5.1 Conclusion

This section includes some applications of a 4-Qubit chip along with potentials for further optimization of the chip.

5.1.1 Application of a 4-Qubit Chip

A 4-qubit superconducting quantum chip serves as a fundamental building block, as well as a practical testbed for scalable systems. Many commercial quantum computing companies, including IBM, Rigetti, and Google, have utilized smaller multi-qubit architectures to benchmark small-scale quantum systems and validate their methodologies before moving to larger-scale implementations. Additionally, a 4-qubit system is large enough to execute fundamental quantum algorithms such as Grover's Search, the Quantum Fourier Transform (QFT), and Variational Quantum Eigensolver (VQE)—providing insights into quantum advantage. This chip size also facilitates research into qubit connectivity and coupling, allowing different topologies (e.g., linear, star, or fully connected) to be tested, optimizing gate execution and mitigating crosstalk effects. Furthermore, in recent works, physical qubits in superconducting systems have reached the point where errors are at or below the threshold, and networks of 4-9 superconducting qubits with individual control and readout have been used to show concepts of error

correction [40, 42, 48].

Application 1:A Basic Entanglement Swapping Circuit

In quantum networks, we need to transmit entanglement over long distances. Since entanglement decoheres over distance, quantum repeaters perform entanglement swapping at intermediate nodes to extend the range of entanglement without sending fragile quantum states directly. This is the core principle behind a quantum repeater. In Figure 5.1, an entanglement swapping circuit using 4 qubits is shown.

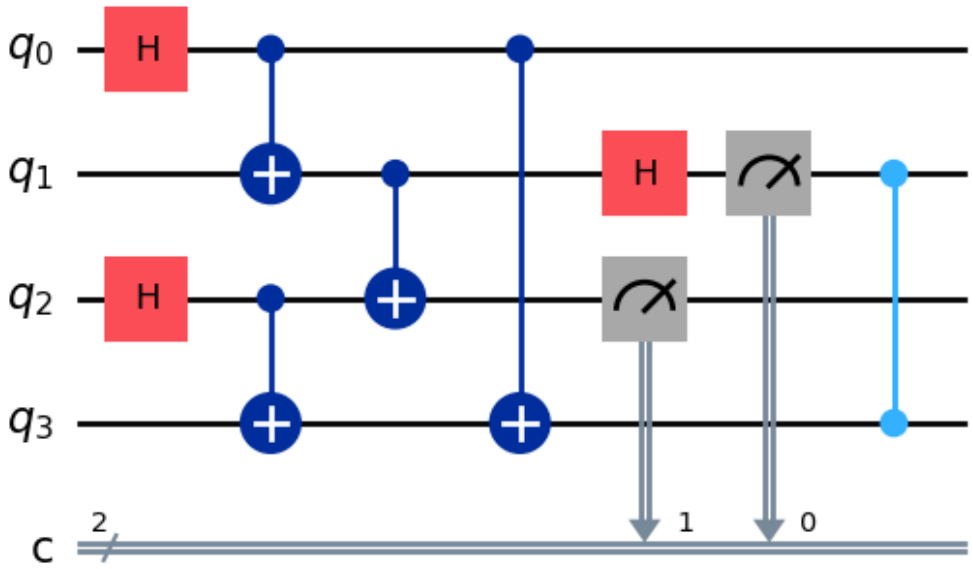


Figure 5.1: A Quantum circuit consisting 4 qubits. Initially, (Q0-Q1) and (Q2-Q3) were entangled separately. After entanglement swapping, Q1 and Q2 are measured, which transfers entanglement to Q0 and Q3. Q0 and Q3 are now entangled, though they are not directly connected

Application 2: Quantum Error Detection Code (Bit-Flip Code)

The goal of an error detection circuit is to encode information in such a way that we can detect (and potentially correct) errors caused by decoherence or noise. The simplest quantum error detection code is the bit-flip code, which encodes one logical qubit into multiple physical qubits. In Figure 5.1, an error detection circuit using 4 qubits is shown.

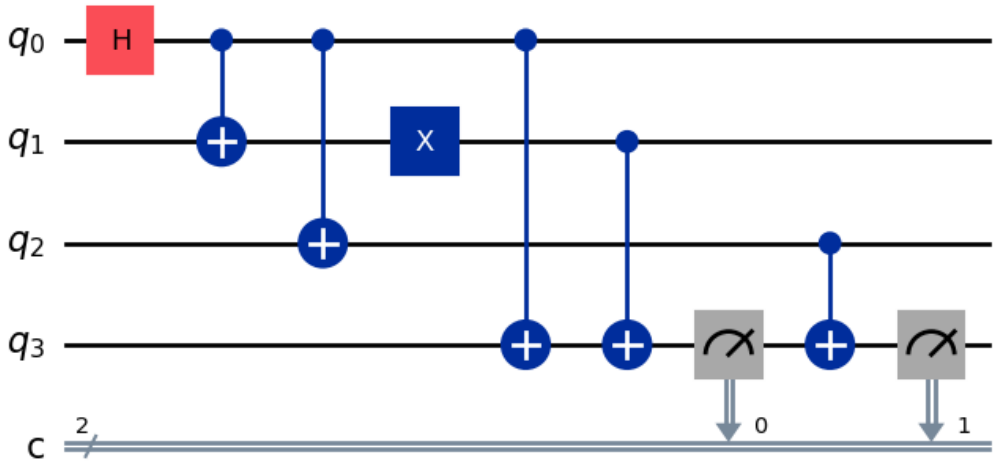


Figure 5.2: A Quantum circuit consisting 4 qubits implementing a quantum error detection code, specifically a bit-flip code. It can detect single-qubit bit-flip errors using syndrome measurements. If an error is detected, additional correction steps could be applied.

5.1.2 Further Optimization of Superconducting Chips

This research was based on design and tuning of a superconducting quantum chip. To further optimize the design and tuning to achieve better performance more rigorous electromagnetic and microwave simulation is needed. Due to heavy computational cost, the simulation of the whole chip in chapter 4 was limited to a few number of passes demonstrating poor convergence. Further optimization can be done incorporating a machine learning method for optimization and tuning.

Electromagnetic and Microwave Simulation Optimization

To ensure optimal qubit performance, full-wave simulations and eigenmode analysis are employed to prevent frequency crowding and minimize crosstalk.

- **Eigenmode and S-Parameter Analysis**: Ansys HFSS simulations help identify mode distributions and impedance matching.
- **Parasitic Element Reduction**: Using Q3D Extractor minimizes stray capacitance and inductance.
- **High-Q Resonator Design**: Coplanar waveguide (CPW) and microstrip adjustments increase quality factors.

- **Choosing different substrate:** In our simulation sillicon was chose as substrate with effective dielectric constant = 11.45. Choice of substrate can help optimize and gain better results.

Machine Learning Optimization for Tuning

Machine learning (ML) techniques, such as reinforcement learning and Bayesian optimization, can enhance qubit parameter tuning and stability.

- Reinforcement Learning for Gate Calibration: ML models optimize qubit frequency tuning and detect frequency collisions [73].
- - Quantum Kernel Methods: ML-based noise classification aids in decoherence mitigation.
- Closed-Loop Control Systems: Real-time feedback mechanisms adjust qubit parameters dynamically [73].

Shanto et al. [74] in 2024 presented an open-source database of superconducting quantum device designs named **SQUAADS**, programmatically generated using Qiskit Metal and validated through simulations and experiments, providing a user-friendly interface to facilitate high-accuracy design customization and lower the barrier for new research.

5.2 Novelty of the Work

This study introduces two innovative designs for 4-qubit superconducting quantum chips. While small-scale superconducting quantum chips with Transmon qubits already exist, the proposed circuits stand out due to their unique circuit geometry. Furthermore, the inclusion of computational cost analysis for tuning qubit and resonator simulations sets this work apart from previous research.

5.3 Scope for Further Research and Future of Superconducting Quantum Chip

Recent advancements in superconducting quantum computing, such as Google's Willow processor with 105 qubits and IBM's cloud-based Quantum Platform, demonstrate rapid progress in scalability and error correction [30]. Microsoft's Majorana 1 chip further enhances qubit

stability using topological superconductors, advancing the field toward fault-tolerant quantum computing [31]. Zuchongzhi 3.0, revealed in March 3, 2025, is a 105-qubit superconducting quantum computer prototype that achieves high operational fidelities, with single-qubit gates, two-qubit gates, and readout fidelity at 99.90%, 99.62%, and 99.13%, respectively [32]. Amazon's Ocelot, released in February 2025, represents an important step on the road to practical quantum computers, leveraging chip-scale integration of cat qubits to form a scalable, hardware-efficient architecture for quantum error correction [33]. With ongoing improvements in fabrication, integrating larger qubit systems is becoming feasible, pushing quantum supremacy closer to reality. Tech giants like Google, IBM, Amazon, and Microsoft are heavily investing in commercialization, with applications in cryptography, drug discovery, and complex modeling. Resources like IBM's Qiskit and Quantum Platform provide researchers with tools to develop quantum algorithms. As coherence and error rates improve, superconducting quantum computing is set to revolutionize next-generation computational technologies. A roadmap A roadmap illustrating the projected advancements in quantum computing across different technologies spanning from 2022 to beyond 2040 is shown in Figure 5.3.

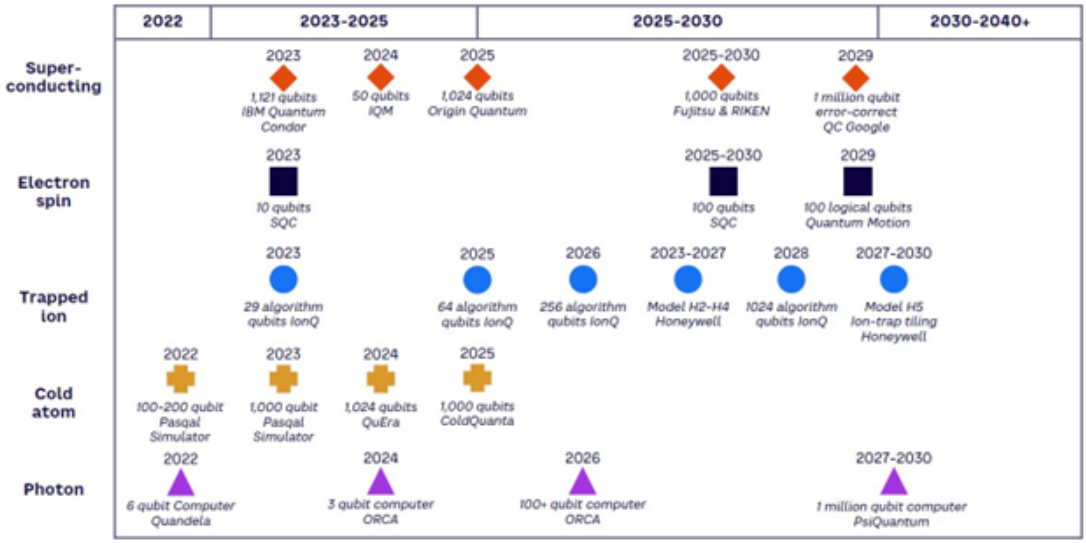


Figure 5.3: A roadmap illustrating the projected advancements in quantum computing across different technologies spanning from 2022 to beyond 2040.

The roadmap depicts that in superconducting qubits, IBM, IQM, and Origin Quantum are scaling their systems, with Google aiming for a 1-million-qubit error-corrected quantum computer by 2029. Electron spin-based quantum computing is progressing from 10 qubits in 2023 (SQC) to 100 logical qubits by 2029 (Quantum Motion). Trapped ion systems, led by IonQ and Honeywell, show a steady increase in qubit counts, targeting 1024 algorithmic qubits by 2028 and multi-qubit tiling models by 2030. Cold atom platforms, including Pasqal, QuEra, and ColdQuanta, project systems exceeding 1000 qubits by 2025. Meanwhile, photon-based quantum computing, represented by ORCA and PsiQuantum, is advancing toward 100+ qubits by

2026, with PsiQuantum aiming for a 1-million-qubit machine by 2030-2040. This roadmap highlights the rapid progress and competition among different quantum computing technologies, emphasizing scalability, error correction, and the path toward fault-tolerant quantum computation. In this paper, the design and tuning of a 4-qubit superconducting quantum chip were explored, comparing direct qubit coupling with a bus resonator architecture to evaluate their impact on qubit performance, connectivity, and coherence. The findings highlight the importance of geometric parameter optimization, resonator tuning, and loss mitigation strategies, paving the way for future improvements in scalability, error correction, and quantum gate fidelity for more robust superconducting quantum processors.

References

[1] Nielsen, M.A. and Chuang, I.L. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010

[2] Preskill, J. "Quantum computing in the nisq era and beyond." *Quantum*, volume 2:p. 79, August 2018

[3] Shor, P.W. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." *SIAM Journal on Computing*, volume 26, no. 5:pp. 1484–1509, 1997

[4] Grover, L.K. "A fast quantum mechanical algorithm for database search." In "Proceedings of the twenty-eighth annual ACM symposium on Theory of computing," pp. 212–219, 1996

[5] Farhi, E., Goldstone, J. and Gutmann, S. "A quantum approximate optimization algorithm.", 2014

[6] Aspuru-Guzik, A., Dutoi, A.D., Love, P.J. and Head-Gordon, M. "Simulated quantum computation of molecular energies." *Science*, volume 309, no. 5741:p. 1704–1707, September 2005

[7] Bernien, H., Schwartz, S., Keesling, A., Levine, H., Omran, A., Pichler, H., Choi, S., Zibrov, A.S., Endres, M., Greiner, M., Vuletić, V. and Lukin, M.D. "Probing many-body dynamics on a 51-atom quantum simulator." *Nature*, volume 551, no. 7682:pp. 579–584, Nov 2017

[8] Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G.S.L., Buell, D.A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., Fowler, A., Gidney, C., Giustina, M., Graff, R., Guerin, K., Habegger, S., Harrigan, M.P., Hartmann, M.J., Ho, A., Hoffmann, M., Huang, T., Humble, T.S., Isakov, S.V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Kelly, J., Klimov, P.V., Knysh, S., Korotkov, A., Kostritsa, F., Landhuis, D., Lindmark, M., Lucero, E., Lyakh, D., Mandrà, S., McClean, J.R., McEwen, M., Megrant, A., Mi, X.,

Michielsen, K., Mohseni, M., Mutus, J., Naaman, O., Neeley, M., Neill, C., Niu, M.Y., Ostby, E., Petukhov, A., Platt, J.C., Quintana, C., Rieffel, E.G., Roushan, P., Rubin, N.C., Sank, D., Satzinger, K.J., Smelyanskiy, V., Sung, K.J., Trevithick, M.D., Vainsencher, A., Villalonga, B., White, T., Yao, Z.J., Yeh, P., Zalcman, A., Neven, H. and Martinis, J.M. "Quantum supremacy using a programmable superconducting processor." *Nature*, volume 574, no. 7779:pp. 505–510, Oct 2019

[9] DiVincenzo, D.P. "The physical implementation of quantum computation." *Fortschritte der Physik*, volume 48, no. 9–11:p. 771–783, September 2000

[10] Ladd, T.D., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C. and O'Brien, J.L. "Quantum computers." *Nature*, volume 464, no. 7285:pp. 45–53, Mar 2010

[11] Bernstein, D.J. and Lange, T. "Post-quantum cryptography." *Nature*, volume 549, no. 7671:pp. 188–194, Sep 2017

[12] Feynman, R.P. "Simulating physics with computers." *International Journal of Theoretical Physics*, volume 21, no. 6:pp. 467–488, Jun 1982

[13] Bruzewicz, C.D., Chiaverini, J., McConnell, R. and Sage, J.M. "Trapped-ion quantum computing: Progress and challenges." *Applied Physics Reviews*, volume 6, no. 2, May 2019

[14] Hanson, R., Kouwenhoven, L.P., Petta, J.R., Tarucha, S. and Vandersypen, L.M.K. "Spins in few-electron quantum dots." *Reviews of Modern Physics*, volume 79, no. 4:p. 1217–1265, October 2007

[15] Rudolph, T. "Why i am optimistic about the silicon-photonic route to quantum computing.", 2016

[16] Krantz, P., Kjaergaard, M., Yan, F., Orlando, T.P., Gustavsson, S. and Oliver, W.D. "A quantum engineer's guide to superconducting qubits." *Applied Physics Reviews*, volume 6, no. 2, June 2019

[17] Gu, X., Kockum, A.F., Miranowicz, A., Liu, Y.x. and Nori, F. "Microwave photonics with superconducting quantum circuits." *Physics Reports*, volume 718–719:p. 1–102, November 2017

[18] Nakamura, Y., Pashkin, Y.A. and Tsai, J.S. "Coherent control of macroscopic quantum states in a single-cooper-pair box." *Nature*, volume 398, no. 6730:p. 786–788, April 1999

[19] Little, A.D. and Ezratty, O. "Core qubit technologies mapped by maturity, intensity, and disruption potential.", 2023. Figure

[20] Koch, J., Yu, T.M., Gambetta, J., Houck, A.A., Schuster, D.I., Majer, J., Blais, A., Devoret, M.H., Girvin, S.M. and Schoelkopf, R.J. "Charge-insensitive qubit design derived from the cooper pair box." *Phys. Rev. A*, volume 76:p. 042319, Oct 2007

[21] Manucharyan, V.E., Koch, J., Glazman, L.I. and Devoret, M.H. "Fluxonium: Single cooper-pair circuit free of charge offsets." *Science*, volume 326, no. 5949:p. 113–116, October 2009

[22] Barends, R., Kelly, J., Megrant, A., Sank, D., Jeffrey, E., Chen, Y., Yin, Y., Chiaro, B., Mutus, J., Neill, C., O'Malley, P., Roushan, P., Wenner, J., White, T.C., Cleland, A.N. and Martinis, J.M. "Coherent josephson qubit suitable for scalable quantum integrated circuits." *Physical Review Letters*, volume 111, no. 8, August 2013

[23] Clarke, J. and Wilhelm, F.K. "Superconducting quantum bits." *Nature*, volume 453, no. 7198:pp. 1031–1042, Jun 2008

[24] Fowler, A.G., Mariantoni, M., Martinis, J.M. and Cleland, A.N. "Surface codes: Towards practical large-scale quantum computation." *Physical Review A*, volume 86, no. 3, September 2012

[25] Kjaergaard, M., Schwartz, M.E., Braumüller, J., Krantz, P., Wang, J.I.J., Gustavsson, S. and Oliver, W.D. "Superconducting qubits: Current state of play." *Annual Review of Condensed Matter Physics*, volume 11, no. Volume 11, 2020:pp. 369–395, 2020

[26] Chen, Y., Neill, C., Roushan, P., Leung, N., Fang, M., Barends, R., Kelly, J., Campbell, B., Chen, Z., Chiaro, B., Dunsworth, A., Jeffrey, E., Megrant, A., Mutus, J., O'Malley, P., Quintana, C., Sank, D., Vainsencher, A., Wenner, J., White, T., Geller, M.R., Cleland, A. and Martinis, J.M. "Qubit architecture with high coherence and fast tunable coupling." *Physical Review Letters*, volume 113, no. 22, November 2014

[27] Ma, R., Saxberg, B., Owens, C., Leung, N., Lu, Y., Simon, J. and Schuster, D.I. "A dissipatively stabilized mott insulator of photons." *Nature*, volume 566, no. 7742:p. 51–57, February 2019

[28] Yuan, B., Wang, W., Liu, F., He, H. and Shan, Z. "Comparison of lumped oscillator model and energy participation ratio methods in designing two-dimensional superconducting quantum chips." *Entropy*, volume 24, no. 6, 2022

[29] Devoret, M.H. and Schoelkopf, R.J. "Superconducting circuits for quantum information: An outlook." *Science*, volume 339, no. 6124:pp. 1169–1174, 2013

[30] Acharya, R., Abanin, D.A., Aghababaie-Beni, L., Aleiner, I., Andersen, T.I., Ansmann, M., Arute, F., Arya, K., Asfaw, A., Astrakhantsev, N., Atalaya, J., Babbush, R., Bacon, D., Ballard, B., Bardin, J.C., Bausch, J., Bengtsson, A., Bilmes, A., Blackwell, S., Boixo, S.,

Bortoli, G., Bourassa, A., Bovaird, J., Brill, L., Broughton, M., Browne, D.A., Buchea, B., Buckley, B.B., Buell, D.A., Burger, T., Burkett, B., Bushnell, N., Cabrera, A., Campero, J., Chang, H.S., Chen, Y., Chen, Z., Chiaro, B., Chik, D., Chou, C., Claes, J., Cleland, A.Y., Cogan, J., Collins, R., Conner, P., Courtney, W., Crook, A.L., Curtin, B., Das, S., Davies, A., De Lorenzo, L., Debroy, D.M., Demura, S., Devoret, M., Di Paolo, A., Donohoe, P., Drozdov, I., Dunsworth, A., Earle, C., Edlich, T., Eickbusch, A., Elbag, A.M., Elzouka, M., Erickson, C., Faoro, L., Farhi, E., Ferreira, V.S., Burgos, L.F., Forati, E., Fowler, A.G., Foxen, B., Ganjam, S., Garcia, G., Gasca, R., Genois, É., Giang, W., Gidney, C., Gilboa, D., Gosula, R., Dau, A.G., Graumann, D., Greene, A., Gross, J.A., Habegger, S., Hall, J., Hamilton, M.C., Hansen, M., Harrigan, M.P., Harrington, S.D., Heras, F.J.H., Heslin, S., Heu, P., Higgott, O., Hill, G., Hilton, J., Holland, G., Hong, S., Huang, H.Y., Huff, A., Huggins, W.J., Ioffe, L.B., Isakov, S.V., Iveland, J., Jeffrey, E., Jiang, Z., Jones, C., Jordan, S., Joshi, C., Juhas, P., Kafri, D., Kang, H., Karamlou, A.H., Kechedzhi, K., Kelly, J., Khaire, T., Khattar, T., Khezri, M., Kim, S., Klimov, P.V., Klots, A.R., Kobrin, B., Kohli, P., Korotkov, A.N., Kostritsa, F., Kothari, R., Kozlovskii, B., Kreikebaum, J.M., Kurilovich, V.D., Lacroix, N., Landhuis, D., Lange-Dei, T., Langley, B.W., Laptev, P., Lau, K.M., Le Guevel, L., Ledford, J., Lee, J., Lee, K., Lensky, Y.D., Leon, S., Lester, B.J., Li, W.Y., Li, Y., Lill, A.T., Liu, W., Livingston, W.P., Locharla, A., Lucero, E., Lundahl, D., Lunt, A., Madhuk, S., Malone, F.D., Maloney, A., Mandrà, S., Manyika, J., Martin, L.S., Martin, O., Martin, S., Maxfield, C., McClean, J.R., McEwen, M., Meeks, S., Megrant, A., Mi, X., Miao, K.C., Mieszala, A., Molavi, R., Molina, S., Montazeri, S., Morvan, A., Movassagh, R., Mruczkiewicz, W., Naaman, O., Neeley, M., Neill, C., Nersisyan, A., Neven, H., Newman, M., Ng, J.H., Nguyen, A., Nguyen, M., Ni, C.H., Niu, M.Y., O'Brien, T.E., Oliver, W.D., Opremcak, A., Ottosson, K., Petukhov, A., Pizzuto, A., Platt, J., Potter, R., Pritchard, O., Pryadko, L.P., Quintana, C., Ramachandran, G., Reagor, M.J., Redding, J., Rhodes, D.M., Roberts, G., Rosenberg, E., Rosenfeld, E., Roushan, P., Rubin, N.C., Saei, N., Sank, D., Sankaragomathi, K., Satzinger, K.J., Schurkus, H.F., Schuster, C., Senior, A.W., Shearn, M.J., Shorter, A., Shutty, N., Shvarts, V., Singh, S., Sivak, V., Skruzny, J., Small, S., Smelyanskiy, V., Smith, W.C., Somma, R.D., Springer, S., Sterling, G., Strain, D., Suchard, J., Szasz, A., Sztein, A., Thor, D., Torres, A., Torunbalci, M.M., Vaishnav, A., Vargas, J., Vdovichev, S., Vidal, G., Villalonga, B., Heidweiller, C.V., Waltman, S., Wang, S.X., Ware, B., Weber, K., Weidel, T., White, T., Wong, K., Woo, B.W.K., Xing, C., Yao, Z.J., Yeh, P., Ying, B., Yoo, J., Yosri, N., Young, G., Zalcman, A., Zhang, Y., Zhu, N., Zobrist, N., AI, G.Q. and Collaborators,. "Quantum error correction below the surface code threshold." *Nature*, volume 638, no. 8052:pp. 920–926, Feb 2025

[31] Aghaee, M., Alcaraz Ramirez, A., Alam, Z., Ali, R., Andrzejczuk, M., Antipov, A., Astafev, M., Barzegar, A., Bauer, B., Becker, J., Bhaskar, U.K., Bocharov, A., Boddapati, S., Bohn, D., Bommer, J., Bourdet, L., Bousquet, A., Boutin, S., Casparis, L.,

Chapman, B.J., Chatoor, S., Christensen, A.W., Chua, C., Codd, P., Cole, W., Cooper, P., Corsetti, F., Cui, A., Dalpasso, P., Dehollain, J.P., de Lange, G., de Moor, M., Ekefjärd, A., El Dandachi, T., Estrada Saldaña, J.C., Fallahi, S., Galletti, L., Gardner, G., Govender, D., Griggio, F., Grigoryan, R., Grijalva, S., Gronin, S., Gukelberger, J., Hamdast, M., Hamze, F., Hansen, E.B., Heedt, S., Heidarnia, Z., Herranz Zamorano, J., Ho, S., Holgaard, L., Hornibrook, J., Indrapiromkul, J., Ingerslev, H., Ivancevic, L., Jensen, T., Jhoja, J., Jones, J., Kalashnikov, K.V., Kallaher, R., Kalra, R., Karimi, F., Karzig, T., King, E., Kloster, M.E., Knapp, C., Kocon, D., Koski, J.V., Kostamo, P., Kumar, M., Laeven, T., Larsen, T., Lee, J., Lee, K., Leum, G., Li, K., Lindemann, T., Looij, M., Love, J., Lucas, M., Lutchyn, R., Madsen, M.H., Madulid, N., Malmros, A., Manfra, M., Mantri, D., Markussen, S.B., Martinez, E., Mattila, M., McNeil, R., Mei, A.B., Mishmash, R.V., Mohandas, G., Mollgaard, C., Morgan, T., Moussa, G., Nayak, C., Nielsen, J.H., Nielsen, J.M., Nielsen, W.H.P., Nijholt, B., Nystrom, M., O'Farrell, E., Ohki, T., Otani, K., Paquelet Wütz, B., Pauka, S., Petersson, K., Petit, L., Pikulin, D., Prawiroatmodjo, G., Preiss, F., Puchol Morejon, E., Rajpalke, M., Ranta, C., Rasmussen, K., Razmadze, D., Reentila, O., Reilly, D.J., Ren, Y., Reneris, K., Rouse, R., Sadovskyy, I., Sainiemi, L., Sanlorenzo, I., Schmidgall, E., Sfiligoj, C., Shah, M.B., Simoes, K., Singh, S., Sinha, S., Soerensen, T., Sohr, P., Stankevic, T., Stek, L., Stuppard, E., Suominen, H., Suter, J., Teicher, S., Thiyagarajah, N., Tholapi, R., Thomas, M., Toomey, E., Tracy, J., Turley, M., Upadhyay, S., Urban, I., Van Hoogdalem, K., Van Woerkom, D.J., Viazmitinov, D.V., Vogel, D., Watson, J., Webster, A., Weston, J., Winkler, G.W., Xu, D., Yang, C.K., Yucelen, E., Zeisel, R., Zheng, G., Zilke, J. and Quantum, M.A. "Interferometric single-shot parity measurement in inas–al hybrid devices." *Nature*, volume 638, no. 8051:pp. 651–655, Feb 2025

[32] Gao, D., Fan, D., Zha, C., Bei, J., Cai, G., Cai, J., Cao, S., Chen, F., Chen, J., Chen, K., Chen, X., Chen, X., Chen, Z., Chen, Z., Chen, Z., Chu, W., Deng, H., Deng, Z., Ding, P., Ding, X., Ding, Z., Dong, S., Dong, Y., Fan, B., Fu, Y., Gao, S., Ge, L., Gong, M., Gui, J., Guo, C., Guo, S., Guo, X., Han, L., He, T., Hong, L., Hu, Y., Huang, H.L., Huo, Y.H., Jiang, T., Jiang, Z., Jin, H., Leng, Y., Li, D., Li, D., Li, F., Li, J., Li, J., Li, J., Li, J., Li, N., Li, S., Li, W., Li, Y., Li, Y., Liang, F., Liang, X., Liao, N., Lin, J., Lin, W., Liu, D., Liu, H., Liu, M., Liu, X., Liu, X., Liu, Y., Lou, H., Ma, Y., Meng, L., Mou, H., Nan, K., Nie, B., Nie, M., Ning, J., Niu, L., Peng, W., Qian, H., Rong, H., Rong, T., Shen, H., Shen, Q., Su, H., Su, F., Sun, C., Sun, L., Sun, T., Sun, Y., Tan, Y., Tan, J., Tang, L., Tu, W., Wan, C., Wang, J., Wang, B., Wang, C., Wang, C., Wang, C., Wang, J., Wang, L., Wang, R., Wang, S., Wang, X., Wang, X., Wang, X., Wang, Y., Wei, Z., Wei, J., Wu, D., Wu, G., Wu, J., Wu, S., Wu, Y., Xie, S., Xin, L., Xu, Y., Xue, C., Yan, K., Yang, W., Yang, X., Yang, Y., Ye, Y., Ye, Z., Ying, C., Yu, J., Yu, Q., Yu, W., Zeng, X., Zhan, S., Zhang, F., Zhang, H., Zhang, K., Zhang, P., Zhang, W., Zhang, Y., Zhang, Y., Zhang, L., Zhao, G., Zhao, P., Zhao, X., Zhao, X., Zhao, Y., Zhao, Z., Zheng, L., Zhou, F., Zhou,

L., Zhou, N., Zhou, N., Zhou, S., Zhou, S., Zhou, Z., Zhu, C., Zhu, Q., Zou, G., Zou, H., Zhang, Q., Lu, C.Y., Peng, C.Z., Zhu, X. and Pan, J.W. "Establishing a new benchmark in quantum computational advantage with 105-qubit zuchongzhi 3.0 processor." *Phys. Rev. Lett.*, volume 134:p. 090601, Mar 2025

[33] Putterman, H., Noh, K., Hann, C.T., MacCabe, G.S., Aghaeimeibodi, S., Patel, R.N., Lee, M., Jones, W.M., Moradinejad, H., Rodriguez, R., Mahuli, N., Rose, J., Owens, J.C., Levine, H., Rosenfeld, E., Reinhold, P., Moncelsi, L., Alcid, J.A., Alidoust, N., Arrangoiz-Arriola, P., Barnett, J., Bienias, P., Carson, H.A., Chen, C., Chen, L., Chinkezian, H., Chisholm, E.M., Chou, M.H., Clerk, A., Clifford, A., Cosmic, R., Curiel, A.V., Davis, E., DeLorenzo, L., D'Ewart, J.M., Diky, A., D'Souza, N., Dumitrescu, P.T., Eisenmann, S., Elkhouly, E., Evenbly, G., Fang, M.T., Fang, Y., Fling, M.J., Fon, W., Garcia, G., Gorshkov, A.V., Grant, J.A., Gray, M.J., Grimberg, S., Grimsmo, A.L., Haim, A., Hand, J., He, Y., Hernandez, M., Hover, D., Hung, J.S.C., Hunt, M., Iverson, J., Jarrige, I., Jaskula, J.C., Jiang, L., Kalaee, M., Karabalin, R., Karalekas, P.J., Keller, A.J., Khalajhedayati, A., Kubica, A., Lee, H., Leroux, C., Lieu, S., Ly, V., Madrigal, K.V., Marcaud, G., McCabe, G., Miles, C., Milsted, A., Minguzzi, J., Mishra, A., Mukherjee, B., Naghiloo, M., Oblepias, E., Ortuno, G., Pagdilao, J., Pancotti, N., Panduro, A., Paquette, J.P., Park, M., Peairs, G.A., Perello, D., Peterson, E.C., Ponte, S., Preskill, J., Qiao, J., Refael, G., Resnick, R., Retzker, A., Reyna, O.A., Runyan, M., Ryan, C.A., Sahmoud, A., Sanchez, E., Sanil, R., Sankar, K., Sato, Y., Scaffidi, T., Siavoshi, S., Sivarajah, P., Skogland, T., Su, C.J., Swenson, L.J., Teo, S.M., Tomada, A., Torlai, G., Wollack, E.A., Ye, Y., Zerrudo, J.A., Zhang, K., Brandão, F.G.S.L., Matheny, M.H. and Painter, O. "Hardware-efficient quantum error correction via concatenated bosonic qubits." *Nature*, volume 638, no. 8052:pp. 927–934, Feb 2025

[34] Schoelkopf, R.J. and Girvin, S.M. "Wiring up quantum systems." *Nature*, volume 451, no. 7179:pp. 664–669, Feb 2008

[35] de Leon, N.P., Itoh, K.M., Kim, D., Mehta, K.K., Northup, T.E., Paik, H., Palmer, B.S., Samarth, N., Sangtawesin, S. and Steuerman, D.W. "Materials challenges and opportunities for quantum computing hardware." *Science*, volume 372, no. 6539:p. eabb2823, 2021

[36] Siddiqi, I. "Engineering high-coherence superconducting qubits." *Nature Reviews Materials*, volume 6, no. 10:pp. 875–891, Oct 2021

[37] Saffman, M. "Quantum computing with atomic qubits and rydberg interactions: progress and challenges." *Journal of Physics B: Atomic, Molecular and Optical Physics*, volume 49, no. 20:p. 202001, October 2016

[38] Nayak, C., Simon, S.H., Stern, A., Freedman, M. and Das Sarma, S. "Non-abelian anyons and topological quantum computation." *Rev. Mod. Phys.*, volume 80:pp. 1083–1159, Sep 2008

[39] Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J.M. and Gambetta, J.M. "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets." *Nature*, volume 549, no. 7671:p. 242–246, September 2017

[40] Córcoles, A.D., Magesan, E., Srinivasan, S.J., Cross, A.W., Steffen, M., Gambetta, J.M. and Chow, J.M. "Demonstration of a quantum error detection code using a square lattice of four superconducting qubits." *Nature Communications*, volume 6, no. 1:p. 6979, Apr 2015

[41] Gao, D., Fan, D., Zha, C., Bei, J., Cai, G., Cai, J., Cao, S., Zeng, X., Chen, F., Chen, J., Chen, K., Chen, X., Chen, X., Chen, Z., Chen, Z., Chen, Z., Chu, W., Deng, H., Deng, Z., Ding, P., Ding, X., Ding, Z., Dong, S., Dong, Y., Fan, B., Fu, Y., Gao, S., Ge, L., Gong, M., Gui, J., Guo, C., Guo, S., Guo, X., He, T., Hong, L., Hu, Y., Huang, H.L., Huo, Y.H., Jiang, T., Jiang, Z., Jin, H., Leng, Y., Li, D., Li, D., Li, F., Li, J., Li, J., Li, J., Li, J., Li, N., Li, S., Li, W., Li, Y., Li, Y., Liang, F., Liang, X., Liao, N., Lin, J., Lin, W., Liu, D., Liu, H., Liu, M., Liu, X., Liu, X., Liu, Y., Lou, H., Ma, Y., Meng, L., Mou, H., Nan, K., Nie, B., Nie, M., Ning, J., Niu, L., Peng, W., Qian, H., Rong, H., Rong, T., Shen, H., Shen, Q., Su, H., Su, F., Sun, C., Sun, L., Sun, T., Sun, Y., Tan, Y., Tan, J., Tang, L., Tu, W., Wan, C., Wang, J., Wang, B., Wang, C., Wang, C., Wang, C., Wang, J., Wang, L., Wang, R., Wang, S., Wang, X., Wei, Z., Wei, J., Wu, D., Wu, G., Wu, J., Wu, S., Wu, Y., Xie, S., Xin, L., Xu, Y., Xue, C., Yan, K., Yang, W., Yang, X., Yang, Y., Ye, Y., Ye, Z., Ying, C., Yu, J., Yu, Q., Yu, W., Zhan, S., Zhang, F., Zhang, H., Zhang, K., Zhang, P., Zhang, W., Zhang, Y., Zhang, Y., Zhang, L., Zhao, G., Zhao, P., Zhao, X., Zhao, X., Zhao, Y., Zhao, Z., Zheng, L., Zhou, F., Zhou, L., Zhou, N., Zhou, N., Zhou, S., Zhou, S., Zhou, Z., Zhu, C., Zhu, Q., Zou, G., Zou, H., Zhang, Q., Lu, C.Y., Peng, C.Z., Zhu, X. and Pan, J.W. "Establishing a new benchmark in quantum computational advantage with 105-qubit zuchongzhi 3.0 processor.", 2024

[42] Kelly, J., Barends, R., Fowler, A.G., Megrant, A., Jeffrey, E., White, T.C., Sank, D., Mutus, J.Y., Campbell, B., Chen, Y., Chen, Z., Chiaro, B., Dunsworth, A., Hoi, I.C., Neill, C., O'Malley, P.J.J., Quintana, C., Roushan, P., Vainsencher, A., Wenner, J., Cleland, A.N. and Martinis, J.M. "State preservation by repetitive error detection in a superconducting quantum circuit." *Nature*, volume 519, no. 7541:p. 66–69, March 2015

[43] Sivak, V.V., Eickbusch, A., Royer, B., Singh, S., Tsioutsios, I., Ganjam, S., Miano, A., Brock, B.L., Ding, A.Z., Frunzio, L., Girvin, S.M., Schoelkopf, R.J. and Devoret, M.H. "Real-time quantum error correction beyond break-even." *Nature*, volume 616, no. 7955:p. 50–55, March 2023

[44] Mi, X., Ippoliti, M., Quintana, C., Greene, A., Chen, Z., Gross, J., Arute, F., Arya, K., Atalaya, J., Babbush, R., Bardin, J., Basso, J., Bengtsson, A., Bilmes, A., Bourassa, A., Brill, L., Broughton, M., Buckley, B., Buell, D., Burkett, B., Bushnell, N., Chiaro, B., Collins, R., Courtney, W., Debroy, D., Demura, S., Derk, A., Dunsworth, A., Eppens, D., Erickson, C., Farhi, E., Fowler, A., Foxen, B., Gidney, C., Giustina, M., Harrigan, M., Harrington, S., Hilton, J., Ho, A., Hong, S., Huang, T., Huff, A., Huggins, W., Ioffe, L., Isakov, S., Iveland, J., Jeffrey, E., Jiang, Z., Jones, C., Kafri, D. and Sondhi, S. "Time-crystalline eigenstate order on a quantum processor." *Nature*, volume 601, no. 7894:pp. 531–536, 2021

[45] Satzinger, K.J., Liu, Y.J., Smith, A., Knapp, C., Newman, M., Jones, C., Chen, Z., Quintana, C., Mi, X., Dunsworth, A., Gidney, C., Aleiner, I., Arute, F., Arya, K., Atalaya, J., Babbush, R., Bardin, J.C., Barends, R., Basso, J., Bengtsson, A., Bilmes, A., Broughton, M., Buckley, B.B., Buell, D.A., Burkett, B., Bushnell, N., Chiaro, B., Collins, R., Courtney, W., Demura, S., Derk, A.R., Eppens, D., Erickson, C., Faoro, L., Farhi, E., Fowler, A.G., Foxen, B., Giustina, M., Greene, A., Gross, J.A., Harrigan, M.P., Harrington, S.D., Hilton, J., Hong, S., Huang, T., Huggins, W.J., Ioffe, L.B., Isakov, S.V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Khattar, T., Kim, S., Klimov, P.V., Korotkov, A.N., Kostritsa, F., Landhuis, D., Laptev, P., Locharla, A., Lucero, E., Martin, O., McClean, J.R., McEwen, M., Miao, K.C., Mohseni, M., Montazeri, S., Mruczkiewicz, W., Mutus, J., Naaman, O., Neeley, M., Neill, C., Niu, M.Y., O'Brien, T.E., Opremcak, A., Pató, B., Petukhov, A., Rubin, N.C., Sank, D., Shvarts, V., Strain, D., Szalay, M., Villalonga, B., White, T.C., Yao, Z., Yeh, P., Yoo, J., Zalcman, A., Neven, H., Boixo, S., Megrant, A., Chen, Y., Kelly, J., Smelyanskiy, V., Kitaev, A., Knap, M., Pollmann, F. and Roushan, P. "Realizing topologically ordered states on a quantum processor." *Science*, volume 374, no. 6572:p. 1237–1241, December 2021

[46] Clerk, A.A., Lehnert, K.W., Bertet, P., Petta, J.R. and Nakamura, Y. "Hybrid quantum systems with circuit quantum electrodynamics." *Nature Physics*, volume 16, no. 3:pp. 257–267, Mar 2020

[47] Xiang, Z.L., Ashhab, S., You, J.Q. and Nori, F. "Hybrid quantum circuits: Superconducting circuits interacting with other quantum systems." *Rev. Mod. Phys.*, volume 85:pp. 623–653, Apr 2013

[48] Ristè, D., Poletto, S., Huang, M.Z., Bruno, A., Vesterinen, V., Saira, O.P. and DiCarlo, L. "Detecting bit-flip errors in a logical qubit using stabilizer measurements." *Nature Communications*, volume 6, no. 1, April 2015

[49] Leggett, A.J. "Macroscopic Quantum Systems and the Quantum Theory of Measurement." *Progress of Theoretical Physics Supplement*, volume 69:pp. 80–100, January 1980

[50] Josephson, B. "Possible new effects in superconductive tunnelling." *Physics Letters*, volume 1, no. 7:pp. 251–253, 1962

[51] Sakurai, J. and Napolitano, J. *Modern Quantum Mechanics Pearson New International Edition*. Pearson Deutschland, 2013

[52] Devoret, M.H. "Quantum Fluctuations in Electrical Circuits." In S. Reynaud, E. Giacobino, and J. Zinn-Justin, editors, "Fluctuations Quantiques/Quantum Fluctuations," p. 351, January 1997

[53] Tinkham, M. *Introduction to Superconductivity*. Dover Publications, 2 edition, June 2004

[54] Frisk Kockum, A. and Nori, F. *Quantum Bits with Josephson Junctions*, pp. 703–741, 09 2019

[55] JOSEPHSON, B.D. "Coupled superconductors." *Rev. Mod. Phys.*, volume 36:pp. 216–220, Jan 1964

[56] Hassani, F., Peruzzo, M., Kapoor, L.N., Trioni, A., Zemlicka, M. and Fink, J.M. "Inductively shunted transmons exhibit noise insensitive plasmon states and a fluxon decay exceeding 3 hours." *Nature Communications*, volume 14, no. 1:p. 3968, Jul 2023

[57] Krinner, S., Lacroix, N., Remm, A., Di Paolo, A., Genois, E., Leroux, C., Hellings, C., Lazar, S., Swiadek, F., Herrmann, J., Norris, G.J., Andersen, C.K., Müller, M., Blais, A., Eichler, C. and Wallraff, A. "Realizing repeated quantum error correction in a distance-three surface code." *Nature*, volume 605, no. 7911:p. 669–674, May 2022

[58] Blais, A., Grimsmo, A.L., Girvin, S.M. and Wallraff, A. "Circuit quantum electrodynamics." *Rev. Mod. Phys.*, volume 93:p. 025005, May 2021

[59] Reilly, D.J. "Engineering the quantum-classical interface of solid-state qubits." *npj Quantum Information*, volume 1, no. 1:p. 15011, Oct 2015

[60] Hornibrook, J., Colless, J., Conway Lamb, I., Pauka, S., Lu, H., Gossard, A., Watson, J., Gardner, G., Fallahi, S., Manfra, M. and Reilly, D. "Cryogenic control architecture for large-scale quantum computing." *Physical Review Applied*, volume 3, no. 2, February 2015

[61] Bardin, J.C., Jeffrey, E., Lucero, E., Huang, T., Naaman, O., Barends, R., White, T., Giustina, M., Sank, D., Roushan, P., Arya, K., Chiaro, B., Kelly, J., Chen, J., Burkett, B., Chen, Y., Dunsworth, A., Fowler, A., Foxen, B., Gidney, C., Graff, R., Klimov, P., Mutus, J., McEwen, M., Megrant, A., Neeley, M., Neill, C., Quintana, C., Vainsencher, A., Neven, H. and Martinis, J. "29.1 a 28nm bulk-cmos 4-to-8ghz ¡2mw cryogenic pulse modulator for scalable quantum computing." In "2019 IEEE International Solid- State Circuits Conference - (ISSCC)," IEEE, February 2019

[62] Riwar, R.P., Hosseinkhani, A., Burkhart, L.D., Gao, Y.Y., Schoelkopf, R.J., Glazman, L.I. and Catelani, G. "Normal-metal quasiparticle traps for superconducting qubits." *Phys. Rev. B*, volume 94:p. 104516, Sep 2016

[63] Yan, F., Gustavsson, S., Kamal, A., Birenbaum, J., Sears, A.P., Hover, D., Gudmundsen, T.J., Rosenberg, D., Samach, G., Weber, S., Yoder, J.L., Orlando, T.P., Clarke, J., Kerman, A.J. and Oliver, W.D. "The flux qubit revisited to enhance coherence and reproducibility." *Nature Communications*, volume 7, no. 1, November 2016

[64] Schreier, J.A., Houck, A.A., Koch, J., Schuster, D.I., Johnson, B.R., Chow, J.M., Gambetta, J.M., Majer, J., Frunzio, L., Devoret, M.H., Girvin, S.M. and Schoelkopf, R.J. "Suppressing charge noise decoherence in superconducting charge qubits." *Physical Review B*, volume 77, no. 18, May 2008

[65] Houck, A.A., Schreier, J.A., Johnson, B.R., Chow, J.M., Koch, J., Gambetta, J.M., Schuster, D.I., Frunzio, L., Devoret, M.H., Girvin, S.M. and Schoelkopf, R.J. "Controlling the spontaneous emission of a superconducting transmon qubit." *Physical Review Letters*, volume 101, no. 8, August 2008

[66] Sete, E.A., Martinis, J.M. and Korotkov, A.N. "Quantum theory of a bandpass purcell filter for qubit readout." *Physical Review A*, volume 92, no. 1, July 2015

[67] Minev, Z.K., Leghtas, Z., Mundhada, S.O., Christakis, L., Pop, I.M. and Devoret, M.H. "Energy-participation quantization of josephson circuits." *npj Quantum Information*, volume 7, no. 1:p. 131, Aug 2021

[68] Minev, Z.K., McConkey, T.G., Takita, M., Corcoles, A.D. and Gambetta, J.M. "Circuit quantum electrodynamics (cqed) with modular quasi-lumped models.", 2021

[69] Solgun, F., Abraham, D.W. and DiVincenzo, D.P. "Blackbox quantization of superconducting circuits using exact impedance synthesis." *Phys. Rev. B*, volume 90:p. 134504, Oct 2014

[70] Burkard, G., Koch, R.H. and DiVincenzo, D.P. "Multilevel quantum description of decoherence in superconducting qubits." *Phys. Rev. B*, volume 69:p. 064503, Feb 2004

[71] Place, A.P.M., Rodgers, L.V.H., Mundada, P., Smitham, B.M., Fitzpatrick, M., Leng, Z., Premkumar, A., Bryon, J., Vrajitoarea, A., Sussman, S., Cheng, G., Madhavan, T., Babla, H.K., Le, X.H., Gang, Y., Jäck, B., Gyenis, A., Yao, N., Cava, R.J., de Leon, N.P. and Houck, A.A. "New material platform for superconducting transmon qubits with coherence times exceeding 0.3 milliseconds." *Nature Communications*, volume 12, no. 1, March 2021

[72] Walter, T., Kurpiers, P., Gasparinetti, S., Magnard, P., Potočnik, A., Salathé, Y., Pechal, M., Mondal, M., Oppliger, M., Eichler, C. and Wallraff, A. "Rapid high-fidelity single-shot dispersive readout of superconducting qubits." *Physical Review Applied*, volume 7, no. 5, May 2017

[73] Bukov, M., Day, A.G., Sels, D., Weinberg, P., Polkovnikov, A. and Mehta, P. "Reinforcement learning in different phases of quantum control." *Physical Review X*, volume 8, no. 3, September 2018

[74] Shanto, S., Kuo, A., Miyamoto, C., Zhang, H., Maurya, V., Vlachos, E., Hecht, M., Shum, C.W. and Levenson-Falk, E. "Squadds: A validated design database and simulation workflow for superconducting qubit design." *Quantum*, volume 8:p. 1465, September 2024

Index

Appendix A

Codes

A.1 Chip Design 1

This code is written in Qiskit Metal for the first design of quantum chip...

```
import qiskit_metal as metal
from qiskit_metal import designs, draw
from qiskit_metal import MetalGUI, Dict, open_docs

from qiskit_metal.qlibrary.qubits.transmon_pocket_6 import TransmonPocket6
from qiskit_metal.qlibrary.qubits.transmon_cross_fl import TransmonCrossFL
from qiskit_metal.qlibrary.qubits.transmon_pocket_cl import TransmonPocketCL

from qiskit_metal.qlibrary.tlines.straight_path import RouteStraight
from qiskit_metal.qlibrary.tlines.meandered import RouteMeander
from qiskit_metal.qlibrary.tlines.pathfinder import RoutePathfinder
from qiskit_metal.qlibrary.tlines.anchored_path import RouteAnchors

from qiskit_metal.qlibrary.lumped.cap_n_interdigital import CapNInterdigital
from qiskit_metal.qlibrary.couplers.cap_n_interdigital_tee import CapNInterdigitalTee
from qiskit_metal.qlibrary.couplers.coupled_line_tee import CoupledLineTee

from qiskit_metal.qlibrary.terminations.launchpad_wb import LaunchpadWirebond
from qiskit_metal.qlibrary.terminations.launchpad_wb_coupled import LaunchpadWirebondCoupled

design = metal.designs.DesignPlanar()
gui = metal.MetalGUI(design)

design.overwrite_enabled = True
design.chips.main

# Size of the chip
design.chips.main.size.size_x = '12mm'
design.chips.main.size.size_y = '12mm'

#Define Qubits
Q_1 = TransmonPocketCL(design,'Q_1', options = dict(
    pos_x='-3mm',
    pos_y='-1mm',
    gds_cell_name = 'FakeJunction_01',
```

```
    hfss_inductance = '14nH',
    pad_width = '425 um',
    pocket_height = '650um',
    connection_pads=dict(
        readout = dict(loc_W=-1, loc_H=1, pad_width = '80um', pad_gap = '50um'),
        bus_13 = dict(loc_W=1, loc_H=1, pad_width = '60um', pad_gap = '10um'),
        bus_14 = dict(loc_W=1, loc_H=-1, pad_width = '60um', pad_gap = '10um')
    )
))

Q_2 = TransmonPocketCL(design,'Q_2', options = dict(
    pos_x='3mm',
    pos_y='1mm',
    orientation='180',
    gds_cell_name = 'FakeJunction_01',
    hfss_inductance = '14nH',
    pad_width = '425 um',
    pocket_height = '650um',
    connection_pads=dict(
        readout = dict(loc_W=-1, loc_H=1, pad_width = '80um', pad_gap = '50um'),
        bus_23 = dict(loc_W=1, loc_H=-1, pad_width = '60um', pad_gap = '10um'),
        bus_24 = dict(loc_W=1, loc_H=1, pad_width = '60um', pad_gap = '10um')
    )
))

Q_3 = TransmonCrossFL(design, 'Q_3', options=dict(
    pos_x='0mm',
    pos_y='3mm',
    orientation='180',
    connection_pads=dict(
        bus_13=dict(connector_location='180', claw_length='95um'),
        bus_23=dict(connector_location='90', claw_length='95um'),
        readout=dict(connector_location='-90')
    ),
    fl_options=dict()
))

Q_4 = TransmonCrossFL(design, 'Q_4', options=dict(
    pos_x='0mm',
    pos_y='-3mm',
    orientation='0',
    connection_pads=dict(
        bus_14=dict(connector_location='90', claw_length='95um'),
        bus_24=dict(connector_location='180', claw_length='95um'),
        readout=dict(connector_location='0')
    ),
    fl_options=dict()
))

# Resonator Length Approximation from theory
from qiskit_metal.analyses.em.cpw_calculations import guided_wavelength
def find_resonator_length(frequency, line_width, line_gap, N):
    # frequency in GHz
    # line_width/line_gap in um
    # N -> 2 for lambda/2, 4 for lambda/4
    # substrate dimensions and properties already set (Default dielectric constant value is
    used)


```

```
    [lambdaG, effSqrt, q] = guided_wavelength(frequency*10**9, line_width*10**-6,
                                              line_gap*10**-6, 750*10**-6, 200*10**-9)

    return str(lambdaG/N*10**3) + " mm"

bus_13_length = find_resonator_length(5.8,10,2,2) #bus13
bus_23_length = find_resonator_length(6,10,2,2) #bus23
bus_24_length = find_resonator_length(6.2,10,2,2) #bus24
bus_14_length = find_resonator_length(6.4,10,2,2) #bus14

#Define Coupling Buses
bus_13 = RouteMeander(design, 'Bus_13', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs = Dict(
                                                start_pin = Dict(
                                                    component = 'Q_1',
                                                    pin = 'bus_13'
                                                ),
                                                end_pin = Dict(
                                                    component = 'Q_3',
                                                    pin = 'bus_13'
                                                )
                                            ),
                                            lead = Dict(
                                                start_straight = '125um',
                                                end_straight = '225um'
                                            ),
                                            meander = Dict(
                                                asymmetry = '50um'),
                                                fillet = "99um",
                                                total_length = '10.65mm'

                                        ))

bus_23 = RouteMeander(design, 'Bus_23', options = dict(
                                        hfss_wire_bonds = True,
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_2',
                                                pin = 'bus_23'
                                            ),
                                            end_pin = Dict(
                                                component = 'Q_3',
                                                pin = 'bus_23'
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '225um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '10.3mm'

                                    ))

bus_24 = RouteMeander(design, 'Bus_24', options = dict(
                                        hfss_wire_bonds = True,
```

```
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_2',
                                                pin = 'bus_24'
                                            ),
                                            end_pin = Dict(
                                                component = 'Q_4',
                                                pin = 'bus_24'
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '225um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '9.97mm'

                                    ))

bus_14 = RouteMeander(design, 'Bus_14', options = dict(
                                        hfss_wire_bonds = True,
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_1',
                                                pin = 'bus_14'
                                            ),
                                            end_pin = Dict(
                                                component = 'Q_4',
                                                pin = 'bus_14'
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '225um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '9.65mm'

                                    ))

# Adding jogs to create smooth meandered lines
from collections import OrderedDict
jogs_start = OrderedDict()
jogs_start[0] = ['L', '350um']
jogs_start[1] = ['L', '500um']

jogs_end = OrderedDict()
jogs_end[0] = ['L', '680um']
jogs_end[1] = ['L','700um']

bus_13 = RouteMeander(design, 'Bus_13', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs = Dict(
                                                start_pin = Dict(
                                                    component = 'Q_1',
```

```
                                                    pin = 'bus_13'
                                                ),
                                                end_pin = Dict(
                                                    component = 'Q_3',
                                                    pin = 'bus_13'
                                                )
                                            ),
                                            lead = Dict(
                                                start_straight = '125um',
                                                end_straight = '225um'
                                            ),
                                            meander = Dict(
                                                asymmetry = '50um'),
                                                fillet = "99um",
                                                total_length = '10.65mm'

                                        ))

jogs_start = OrderedDict()
jogs_start[0] = ['R', '350um']
jogs_start[1] = ['R', '700um']

bus_23 = RouteMeander(design, 'Bus_23', options = dict(
                                        hfss_wire_bonds = True,
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_2',
                                                pin = 'bus_23'
                                            ),
                                            end_pin = Dict(
                                                component = 'Q_3',
                                                pin = 'bus_23'
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '225um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '10.3mm'

                                    ))

jogs_start = OrderedDict()
jogs_start[0] = ['R', '350um']
jogs_start[1] = ['R', '700um']

bus_24 = RouteMeander(design, 'Bus_24', options = dict(
                                        hfss_wire_bonds = True,
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_2',
                                                pin = 'bus_24'
                                            ),
                                            end_pin = Dict(
                                                component = 'Q_4',
                                                pin = 'bus_24'
```

```
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '225um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '9.97mm'

                                    ))

jogs_start = OrderedDict()
jogs_start[0] = ['R', '350um']
jogs_start[1] = ['R', '700um']
bus_14 = RouteMeander(design, 'Bus_14', options = dict(
                                        hfss_wire_bonds = True,
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_1',
                                                pin = 'bus_14'
                                            ),
                                            end_pin = Dict(
                                                component = 'Q_4',
                                                pin = 'bus_14'
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '225um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '9.65mm'

                                    ))


#Define Readout Resonators
Readout_1_length = find_resonator_length(6.8,10,6,2) #Readout_Q1
Readout_1_length = find_resonator_length(7,10,6,2) #Readout_Q2
Readout_1_length = find_resonator_length(7.2,10,6,2) #Readout_Q3
Readout_1_length = find_resonator_length(7.4,10,6,2) #Readout_Q4

jogs_start = OrderedDict()
jogs_start[0] = ['R', '350um']
jogs_end[0] = ['R', '300um']

readout_1 = RouteMeander(design, 'Readout_1', options=dict(
                                    hfss_wire_bonds=True,
                                    pin_inputs=Dict(
                                        start_pin=Dict(
                                            component='Q_1',
                                            pin='readout'
                                        ),
                                        end_pin=Dict(
```

```
                                        component='Cap_Readout_Q_1',
                                        pin='south_end'
                                    )
                                ),
                                lead=Dict(
                                    start_straight='125um',
                                    end_straight='125um',
                                    start_jogged_extension=jogs_start,
                                    end_jogged_extension=jogs_end
                                ),
                                meander=Dict(
                                    asymmetry='150um'),
                                    fillet='99um',
                                    total_length='8.95mm'

                            ))



readout_2 = RouteMeander(design, 'Readout_2', options=dict(
                                hfss_wire_bonds=True,
                                pin_inputs=Dict(
                                    start_pin=Dict(
                                        component='Q_2',
                                        pin='readout'
                                    ),
                                    end_pin=Dict(
                                        component='Cap_Readout_Q_2',
                                        pin='south_end'
                                    )
                                ),
                                lead=Dict(
                                    start_straight='125um',
                                    end_straight='125um',
                                    start_jogged_extension=jogs_start,
                                    # end_jogged_extension=jogs_end
                                ),
                                meander=Dict(
                                    asymmetry='150um'),
                                    fillet='99um',
                                    total_length='8.69mm'

                            ))


readout_3 = RouteMeander(design, 'Readout_3', options=dict(
                                hfss_wire_bonds=True,
                                pin_inputs=Dict(
                                    start_pin=Dict(
                                        component='Q_3',
                                        pin='readout'
                                    ),
                                    end_pin=Dict(
                                        component='Cap_Readout_Q_3',
                                        pin='south_end'
                                    )
```

```
                                        ),
                                        lead=Dict(
                                            start_straight='125um',
                                            end_straight='125um',
                                            start_jogged_extension=jogs_start,
                                            # end_jogged_extension=jogs_end
                                        ),
                                        meander=Dict(
                                            asymmetry='150um'),
                                            fillet='99um',
                                            total_length='8.45mm'

                                    ))


readout_4 = RouteMeander(design, 'Readout_4', options=dict(
                                        hfss_wire_bonds=True,
                                        pin_inputs=Dict(
                                            start_pin=Dict(
                                                component='Q_4',
                                                pin='readout'
                                            ),
                                            end_pin=Dict(
                                                component='Cap_Readout_Q_4',
                                                pin='south_end'
                                            )
                                        ),
                                        lead=Dict(
                                            start_straight='125um',
                                            end_straight='125um',
                                            start_jogged_extension=jogs_start,
                                            # end_jogged_extension=jogs_end
                                        ),
                                        meander=Dict(
                                            asymmetry='150um'),
                                            fillet='99um',
                                            total_length='8.22mm'

                                    ))


#Defining Launchpads and coupling capacitors

launch_readout_q_1 = LaunchpadWirebond(design, 'Launch_Readout_Q_1',
                                            options = dict(
                                                pos_x = '-4mm',
                                                pos_y = '4mm',
                                                orientation = '-90'
                                            ))

launch_readout_q_2 = LaunchpadWirebond(design, 'Launch_Readout_Q_2',
                                            options = dict(
                                                pos_x = '4mm',
                                                pos_y = '-4mm',
                                                orientation = '90'
                                            ))

launch_readout_q_3 = LaunchpadWirebond(design, 'Launch_Readout_Q_3',
```

```
                                        options = dict(
                                            pos_x = '4mm',
                                            pos_y = '4mm',
                                            orientation = '-90'
                                        ))
launch_readout_q_4 = LaunchpadWirebond(design, 'Launch_Readout_Q_4',
                                        options = dict(
                                            pos_x = '-4mm',
                                            pos_y = '-4mm',
                                            orientation = '90'
                                        ))


launch_cl_q_1 = LaunchpadWirebond(design, 'Launch_CL_Q_1',
                                    options = dict(
                                        pos_x = '-4mm',
                                        pos_y = '-1mm',
                                        orientation = '0',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))

launch_cl_q_2 = LaunchpadWirebond(design, 'Launch_CL_Q_2',
                                    options = dict(
                                        pos_x = '4mm',
                                        pos_y = '1mm',
                                        orientation = '180',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))


launch_fl_q_3 = LaunchpadWirebond(design, 'Launch_FL_Q_3',
                                    options = dict(
                                        pos_x = '0mm',
                                        pos_y = '4mm',
                                        orientation = '-90',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))

launch_fl_q_4 = LaunchpadWirebond(design, 'Launch_FL_Q_4',
                                    options = dict(
                                        pos_x = '0mm',
                                        pos_y = '-4mm',
                                        orientation = '90',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))


cap_readout_q_1 = CapNInterdigital(design, 'Cap_Readout_Q_1',
                                   options=dict(
                                        pos_x='-4mm',
                                        pos_y='3.8mm',
                                        orientation='0'
                                   ))
```

```
507
508 cap_readout_q_2 = CapNInterdigital(design, 'Cap_Readout_Q_2',
509                                     options=dict(
510                                         pos_x='4mm',
511                                         pos_y='-3.8mm',
512                                         orientation='180'
513                                     ))
514
515 cap_readout_q_3 = CapNInterdigital(design, 'Cap_Readout_Q_3',
516                                     options=dict(
517                                         pos_x='4mm',
518                                         pos_y='3.8mm',
519                                         orientation='0'
520                                     ))
521
522 cap_readout_q_4 = CapNInterdigital(design, 'Cap_Readout_Q_4',
523                                     options=dict(
524                                         pos_x='-4mm',
525                                         pos_y='-3.8mm',
526                                         orientation='180'
527                                     ))
528
529 #Define charge lines and flux lines
530 jogs_start = OrderedDict()
531 jogs_start[0] = ['R', '350um']
532 jogs_end[0] = ['R', '100um']
533
534
535 charge_line_q_1 = RoutePathfinder(design, 'Charge_Line_Q_1', options = dict(
536                                             hfss_wire_bonds = True,
537                                             fillet='99um',
538                                             trace_width = '5um',
539                                             trace_gap = '3um',
540
541                                             lead=dict(
542                                                 start_straight='125um',
543                                                 end_straight='125um',
544                                                 start_jogged_extension=jogs_start,
545                                                 #end_jogged_extension=jogs_end
546                                                 #end_straight='150um'
547                                                 ),
548                                             pin_inputs=Dict(
549                                                 start_pin=Dict(
550                                                     component='Launch_CL_Q_1',
551                                                     pin='tie'),
552                                                 end_pin=Dict(
553                                                     component='Q_1',
554                                                     pin='Charge_Line')
555                                             )))
556
557
558
559
560
561 charge_line_q_2 = RoutePathfinder(design, 'Charge_Line_Q_2', options = dict(
562                                             hfss_wire_bonds = True,
563                                             fillet='99um',
564                                             trace_width = '5um',
565                                             trace_gap = '3um',
```

```
566
567                                             lead=dict(
568                                                 start_straight='125um',
569                                                 end_straight='125um',
570                                                 start_jogged_extension=jogs_start,
571                                             ),
572                                             pin_inputs=Dict(
573                                                 start_pin=Dict(
574                                                     component='Launch_CL_Q_2',
575                                                     pin='tie'),
576                                                 end_pin=Dict(
577                                                     component='Q_2',
578                                                     pin='Charge_Line')
579                                             )))
580
581 flux_line_q_3 = RoutePathfinder(design, 'Flux_Line_Q_3', options = dict(
582                                             hfss_wire_bonds = True,
583                                             pin_inputs=Dict(
584                                                 start_pin=Dict(
585                                                     component='Q_3',
586                                                     pin='flux_line'
587                                                 ),
588                                                 end_pin=Dict(
589                                                     component='Launch_FL_Q_3',
590                                                     pin='tie'
591                                                 )
592                                             ),
593                                             fillet = '99um',
594                                             trace_width = '5um',
595                                             trace_gap = '3um'
596                                         ))
597 flux_line_q_4 = RoutePathfinder(design, 'Flux_Line_Q_4', options = dict(
598                                             hfss_wire_bonds = True,
599                                             pin_inputs=Dict(
600                                                 start_pin=Dict(
601                                                     component='Q_4',
602                                                     pin='flux_line'
603                                                 ),
604                                                 end_pin=Dict(
605                                                     component='Launch_FL_Q_4',
606                                                     pin='tie'
607                                                 )
608                                             ),
609                                             fillet = '99um',
610                                             trace_width = '5um',
611                                             trace_gap = '3um'
612                                         ))
613
614 #Define transmission lines
615 tl_readout_q_1 = RouteStraight(design, 'TL_Readout_Q_1', options = dict(
616                                             hfss_wire_bonds = True,
617                                             pin_inputs=Dict(
618                                                 start_pin=Dict(
619                                                     component='Cap_Readout_Q_1',
620                                                     pin='north_end'
621                                                 ),
622                                                 end_pin=Dict(
623                                                     component='Launch_Readout_Q_1',
624                                                     pin='tie'
```

```
                                                )
                                            ),
                                            trace_width = '10um',
                                            trace_gap = '6um'
                                        ))
tl_readout_q_2 = RouteStraight(design, 'TL_Readout_Q_2', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
                                                start_pin=Dict(
                                                    component='Cap_Readout_Q_2',
                                                    pin='north_end'
                                                ),
                                                end_pin=Dict(
                                                    component='Launch_Readout_Q_2',
                                                    pin='tie'
                                                )
                                            ),
                                            trace_width = '10um',
                                            trace_gap = '6um'
                                        ))

tl_readout_q_3 = RouteStraight(design, 'TL_Readout_Q_3', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
                                                start_pin=Dict(
                                                    component='Cap_Readout_Q_3',
                                                    pin='north_end'
                                                ),
                                                end_pin=Dict(
                                                    component='Launch_Readout_Q_3',
                                                    pin='tie'
                                                )
                                            ),
                                            trace_width = '10um',
                                            trace_gap = '6um'
                                        ))

tl_readout_q_4 = RouteStraight(design, 'TL_Readout_Q_4', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
                                                start_pin=Dict(
                                                    component='Cap_Readout_Q_4',
                                                    pin='north_end'
                                                ),
                                                end_pin=Dict(
                                                    component='Launch_Readout_Q_4',
                                                    pin='tie'
                                                )
                                            ),
                                            trace_width = '10um',
                                            trace_gap = '6um'
                                        ))

gui.rebuild()
gui.autoscale()

#Taking Screeshot
gui.screenshot()
```

A.2 Tuning of Chip 1

This is a Qiskit Metal code for tuning the first design of quantum chip. To avoid lengthy code section, tuning process of only 1 qubit and 1 resonator is shown...

```
#Tuning the Qubits using LOM

#Qubit 1
from qiskit_metal.analyses.quantization import LOManalysis

#Rendering using Ansys Q3D
c1 = LOManalysis(design, "q3d")
c1.sim.run(name="Q_1", components=['Q_1'], open_terminations=[('Q_1', 'readout'), ('Q_1', '
    bus_13'),('Q_1', 'bus_14'), ('Q_1', 'Charge_Line')])

#Extracting Capacitance Matrix
c1.sim.capacitance_matrix

c1.setup.junctions = Dict({'Lj': 14.9, 'Cj': 2})
c1.setup.freq_readout = 6.8
c1.setup.freq_bus = [5.8, 6, 8]

c1.run_lom()
c1.lumped_oscillator_all

# Plotting Convergence graphs

c1.plot_convergence()
c1.plot_convergence_chi()

#Closing design window
c1.sim.renderer.clean_active_design()


#Run simulations until best tuning parameters are found
#Rest of the qubits are tuned in similar manner. To avoid length of this thesis book, rest of
    the codes are not shared here

#Tuning the resonators using EPR
from qiskit_metal.analyses.quantization import EPRanalysis
eig_res = EPRanalysis(design, "hfss")

bus_13_hfss = eig_res.sim.renderer
bus_13_hfss.start()

bus_13_hfss.activate_ansys_design("Resonator_tuning", 'eigenmode')  # use new_ansys_design()
    to force creation of a blank design

bus_13_hfss.render_design(selection = ['Bus_13', 'Q_1', 'Q_3'], ignored_jjs = [('Q_1', '
    rect_jj'), ('Q_3', 'rect_jj')], box_plus_buffer = False)

bus_13_setup = bus_13_hfss.pinfo.setup
bus_13_setup.passes = 12
bus_13_setup.n_modes = 2

print(f"""
Number of eigenmodes to find             = {bus_13_setup.n_modes}
Number of simulation passes              = {bus_13_setup.passes}
```

```
Convergence freq max delta percent diff  = {bus_13_setup.delta_f}
""")

# Next 2 lines are counterinuitive, since there is no junction in this resonator.
# However, these are necessary to make pyEPR work correctly. Please do note delete
bus_13_hfss.pinfo.design.set_variable('Lj', '10 nH')
bus_13_hfss.pinfo.design.set_variable('Cj', '0 fF')

bus_13_setup.analyze()

bus_13_setup.get_solutions().eigenmodes()

#Plotting Convergence
eig_res.sim.convergence_t, eig_res.sim.convergence_f, _ = bus_13_hfss.get_convergences()
eig_res.sim.plot_convergences()
bus_13_hfss.modeler._modeler.ShowWindow()

#Ploting Electric Fields
bus_13_hfss.plot_fields('main')

# Changing variable and redo the whole process
bus_13.options.total_length = '10.5mm'
gui.rebuild()
bus_13_hfss.clean_active_design()

#Rest of the buses are tuned in the similar manner
```

A.3 Tuning Verification of Chip 1

This is a Qiskit Metal code for verification of tuning the first design of quantum chip...

```
#updating the parameters
Q_1.options.pad_gap = '28um'
Q_1.options.connection_pads.readout.pad_gap = '15um'
Q_1.options.connection_pads.readout.pad_width = '150um'
Q_1.options.connection_pads.bus_13.pad_width = '130um'
Q_1.options.connection_pads.bus_14.pad_width = '110um'
Q_1.options.hfss_inductance = '14.9nH'
Q_1.options.hfss_capacitance = '2fF'

Q_2.options.pad_gap = '28um'
Q_2.options.connection_pads.readout.pad_gap = '15um'
Q_2.options.connection_pads.readout.pad_width = '130um'
Q_2.options.connection_pads.bus_23.pad_width = '100um'
Q_2.options.connection_pads.bus_24.pad_width = '110um'
Q_2.options.hfss_inductance = '13.3nH'
Q_2.options.hfss_capacitance = '4fF'

Q_3.options.cross_width = '18um'
Q_3.options.connection_pads.bus_13.claw_width = '20um'
Q_3.options.connection_pads.bus_13.claw_length = '120um'
Q_3.options.connection_pads.bus_23.claw_width = '20um'
Q_3.options.connection_pads.bus_23.claw_length = '120um'
Q_3.options.hfss_inductance = '8nH'
Q_3.options.hfss_capacitance = '3fF'

```

```
Q_4.options.cross_width = '18um'
Q_4.options.connection_pads.bus_14.claw_width = '20um'
Q_4.options.connection_pads.bus_14.claw_length = '120um'
Q_4.options.connection_pads.bus_24.claw_width = '20um'
Q_4.options.connection_pads.bus_24.claw_length = '120um'
Q_4.options.hfss_inductance = '7.5nH'
Q_4.options.hfss_capacitance = '3fF'

bus_13.options.total_length = '9.5mm'
bus_23.options.total_length = '9.2mm'
bus_24.options.total_length = '9mm'
bus_14.options.total_length = '8.5mm'

bus_13.options.fillet = '50um'
bus_24.options.fillet = '50um'

readout_1.options.total_length = '8.3mm'
readout_2.options.total_length = '8mm'
readout_3.options.total_length = '7.9mm'
readout_4.options.total_length = '7.7mm'


gui.rebuild()

import pyEPR as epr
from qiskit_metal.analyses.quantization import EPRanalysis

# TODO: fold this inside either an analysis class method, or inside the analysis class setup
qcomps = design.components  # short handle (alias)
qcomps['Q_1'].options['hfss_inductance'] = 'Lj1'
qcomps['Q_1'].options['hfss_capacitance'] = 'Cj1'
qcomps['Q_2'].options['hfss_inductance'] = 'Lj2'
qcomps['Q_2'].options['hfss_capacitance'] = 'Cj2'
qcomps['Q_3'].options['hfss_inductance'] = 'Lj3'
qcomps['Q_3'].options['hfss_capacitance'] = 'Cj3'
qcomps['Q_4'].options['hfss_inductance'] = 'Lj4'
qcomps['Q_4'].options['hfss_capacitance'] = 'Cj4'


gui.rebuild()  # line needed to propagate the updates from the qubit instance into the
    junction design table
gui.autoscale()

eig_2qb = EPRanalysis(design, "hfss")

eig_2qb.sim.setup.max_passes = 10
eig_2qb.sim.setup.max_delta_f = 0.05
eig_2qb.sim.setup.n_modes = 12
eig_2qb.sim.setup.vars = Dict(Lj1= '14.9 nH', Cj1= '2 fF',
                              Lj2= '13.3 nH', Cj2= '4 fF',
                              Lj3= '7.5 nH', Cj3= '3 fF',
                              Lj4= '7.5 nH', Cj4= '3 fF')
eig_2qb.sim.setup

# TODO: fold this inside either an analysis class method, or inside the analysis class setup

eig_2qb.sim.renderer.options['x_buffer_width_mm'] = 0.5
eig_2qb.sim.renderer.options['y_buffer_width_mm'] = 0.5
eig_2qb.sim.renderer.options
```

```

#Measuring Comutational Complexity while rendering
import time
start = time.time()
print("Counting")

eig_2qb.sim.run(name="ChipWithoutReadout",
                components=['Q_1', 'Q_2','Q_3','Q_4','Bus_13','Bus_23','Bus_24','Bus_14','Readout_1', 'Readout_2', 'Readout_3', 'Readout_4',
                            'Launch_Readout_Q_1','Launch_Readout_Q_2','Launch_Readout_Q_3','Launch_Readout_Q_4',
                            'Launch_CL_Q_1','Launch_CL_Q_2','Launch_FL_Q_3','Launch_FL_Q_4',
                             'Cap_Readout_Q_1','Cap_Readout_Q_2','Cap_Readout_Q_3','Cap_Readout_Q_4',
                            'Charge_Line_Q_1','Charge_Line_Q_2','Flux_Line_Q_3','Flux_Line_Q_4',
                            'TL_Readout_Q_1','TL_Readout_Q_2','TL_Readout_Q_3','TL_Readout_Q_4'])

end = time.time()
print("Time Elapsed:", end - start)

eig_2qb.sim.plot_convergences()

eig_2qb.get_frequencies()

eig_2qb.setup.junctions.jj1 = Dict(rect='JJ_rect_Lj_Q_1_rect_jj', line='JJ_Lj_Q_1_rect_jj_',
                   Lj_variable='Lj1', Cj_variable='Cj1')
eig_2qb.setup.junctions.jj2 = Dict(rect='JJ_rect_Lj_Q_2_rect_jj', line='JJ_Lj_Q_2_rect_jj_',
                   Lj_variable='Lj2', Cj_variable='Cj2')
eig_2qb.setup.junctions.jj3 = Dict(rect='JJ_rect_Lj_Q_3_rect_jj', line='JJ_Lj_Q_3_rect_jj_',
                   Lj_variable='Lj3', Cj_variable='Cj3')
eig_2qb.setup.junctions.jj4 = Dict(rect='JJ_rect_Lj_Q_4_rect_jj', line='JJ_Lj_Q_4_rect_jj_',
                   Lj_variable='Lj4', Cj_variable='Cj4')



eig_2qb.setup.sweep_variable = 'Lj1'
eig_2qb.setup

# Running EPR analysis to get the energy participation ratio
eig_2qb.run_epr()

eig_2qb.sim.close()
```

A.4 GDS rendering of Chip 1

This is a Qiskit Metal code for GDS rendering of the first design of quantum chip.. . .

```
#GDS Rendering

mt_gds = design.renderers.gds
mt_gds.options['path_filename'] = '../qiskit-metal-main/qiskit-metal-main/tutorials/resources/Fake_Junctions.GDS'


```

```

mt_gds.options.no_cheese.buffer = '40um'
mt_gds.options.max_points = 199
mt_gds.options.cheese.cheese_0_x = '2um'
mt_gds.options.cheese.cheese_0_y = '1um'
mt_gds.options.cheese.view_in_file = {'main': {1: True}}


mt_gds.export_to_gds("MyDesign1.gds")

#Open KLayout to view renderd gds file
```

A.5 Chip Design 2

This code is written in Qiskit Metal for the second design of quantum chip...

```
import qiskit_metal as metal
from qiskit_metal import designs, draw
from qiskit_metal import MetalGUI, Dict, open_docs

from qiskit_metal.qlibrary.qubits.transmon_pocket_6 import TransmonPocket6
from qiskit_metal.qlibrary.qubits.transmon_cross_fl import TransmonCrossFL
from qiskit_metal.qlibrary.qubits.transmon_pocket_cl import TransmonPocketCL

from qiskit_metal.qlibrary.tlines.straight_path import RouteStraight
from qiskit_metal.qlibrary.tlines.meandered import RouteMeander
from qiskit_metal.qlibrary.tlines.pathfinder import RoutePathfinder
from qiskit_metal.qlibrary.tlines.anchored_path import RouteAnchors

from qiskit_metal.qlibrary.lumped.cap_n_interdigital import CapNInterdigital
from qiskit_metal.qlibrary.couplers.cap_n_interdigital_tee import CapNInterdigitalTee
from qiskit_metal.qlibrary.couplers.coupled_line_tee import CoupledLineTee

from qiskit_metal.qlibrary.terminations.launchpad_wb import LaunchpadWirebond
from qiskit_metal.qlibrary.terminations.launchpad_wb_coupled import LaunchpadWirebondCoupled

design = metal.designs.DesignPlanar()
gui = metal.MetalGUI(design)
design.overwrite_enabled = True
design.chips.main

design.chips.main.size.size_x = '10mm'
design.chips.main.size.size_y = '10mm'

Q_1 = TransmonPocketCL(design,'Q_1', options = dict(
    pos_x='-3mm',
    pos_y='2mm',
    gds_cell_name = 'FakeJunction_01',
    hfss_inductance = '14nH',
    pad_width = '425 um',
    pocket_height = '650um',
    connection_pads=dict(
        readout = dict(loc_W=1, loc_H=1, pad_width = '80um', pad_gap = '50um'),
        bus_1 = dict(loc_W=1, loc_H=-1, pad_width = '60um', pad_gap = '10um')

    )
```

```
))

Q_2 = TransmonPocketCL(design,'Q_2', options = dict(
    pos_x='3mm',
    pos_y='-2mm',
    orientation='180',
    gds_cell_name = 'FakeJunction_01',
    hfss_inductance = '14nH',
    pad_width = '425 um',
    pocket_height = '650um',
    connection_pads=dict(
        readout = dict(loc_W=1, loc_H=1, pad_width = '80um', pad_gap = '50um'),
        bus_2 = dict(loc_W=1, loc_H=-1, pad_width = '60um', pad_gap = '10um')

    )
))

Q_3 = TransmonCrossFL(design, 'Q_3', options=dict(
    pos_x='-1mm',
    pos_y='-2mm',
    orientation='0',
    connection_pads=dict(
        bus_3=dict(connector_location='90', claw_length='95um'),
        readout=dict(connector_location='0')
    ),
    fl_options=dict()
))

Q_4 = TransmonCrossFL(design, 'Q_4', options=dict(
    pos_x='1mm',
    pos_y='2mm',
    orientation='180',
    connection_pads=dict(
        bus_4=dict(connector_location='90', claw_length='95um'),
        readout=dict(connector_location='0')
    ),
    fl_options=dict()
))

gui.rebuild()
gui.autoscale()

launch_readout_q_1 = LaunchpadWirebond(design, 'Launch_Readout_Q_1',
                                       options = dict(
                                           pos_x = '-3mm',
                                           pos_y = '5mm',
                                           orientation = '-90'
                                       ))

launch_readout_q_2 = LaunchpadWirebond(design, 'Launch_Readout_Q_2',
                                       options = dict(
                                           pos_x = '3mm',
                                           pos_y = '-5mm',
                                           orientation = '90'
                                       ))

launch_readout_q_3 = LaunchpadWirebond(design, 'Launch_Readout_Q_3',
                                       options = dict(
                                           pos_x = '-5mm',
```

```
                                            pos_y = '-2mm',
                                            orientation = '0'
                                        ))
launch_readout_q_4 = LaunchpadWirebond(design, 'Launch_Readout_Q_4',
                                        options = dict(
                                            pos_x = '5mm',
                                            pos_y = '2mm',
                                            orientation = '180'
                                        ))



gui.rebuild()
gui.autoscale()

launch_cl_q_1 = LaunchpadWirebond(design, 'Launch_CL_Q_1',
                                    options = dict(
                                        pos_x = '-5mm',
                                        pos_y = '2mm',
                                        orientation = '0',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))

launch_cl_q_2 = LaunchpadWirebond(design, 'Launch_CL_Q_2',
                                    options = dict(
                                        pos_x = '5mm',
                                        pos_y = '-2mm',
                                        orientation = '180',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))


launch_fl_q_3 = LaunchpadWirebond(design, 'Launch_FL_Q_3',
                                    options = dict(
                                        pos_x = '-1mm',
                                        pos_y = '-5mm',
                                        orientation = '90',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))

launch_fl_q_4 = LaunchpadWirebond(design, 'Launch_FL_Q_4',
                                    options = dict(
                                        pos_x = '1mm',
                                        pos_y = '5mm',
                                        orientation = '-90',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))



gui.rebuild()
gui.autoscale()

launch_bus_main_1 = LaunchpadWirebond(design, 'Launch_BUS_MAIN_1',
```

```
                                    options = dict(
                                        pos_x = '-5mm',
                                        pos_y = '0mm',
                                        orientation = '0',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))

launch_bus_main_2 = LaunchpadWirebond(design, 'Launch_BUS_MAIN_2',
                                    options = dict(
                                        pos_x = '5mm',
                                        pos_y = '0mm',
                                        orientation = '180',
                                        trace_width = '5um',
                                        trace_gap = '3um'
                                    ))

gui.rebuild()
gui.autoscale()

bus_main = RouteStraight(design, 'Bus_main', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
                                                start_pin=Dict(
                                                    component='Launch_BUS_MAIN_1',
                                                    pin='tie'),
                                                end_pin=Dict(
                                                    component='Launch_BUS_MAIN_2',
                                                    pin='tie')),
                                                trace_width = '10um',
                                                trace_gap = '6um'
                                                ))

gui.rebuild()
gui.autoscale()

cap_couple_q_1 = CapNInterdigitalTee(design, 'Cap_Couple_Q_1',
                                    options=dict(
                                        pos_x='-3mm',
                                        pos_y='0mm',
                                        orientation='180'
                                    ))

cap_couple_q_2 = CapNInterdigitalTee(design, 'Cap_Couple_Q_2',
                                    options=dict(
                                        pos_x='3mm',
                                        pos_y='0mm',
                                        orientation='0'
                                    ))

cap_couple_q_3 = CapNInterdigitalTee(design, 'Cap_Couple_Q_3',
                                    options=dict(
                                        pos_x='-1mm',
                                        pos_y='0mm',
                                        orientation='0'
                                    ))

cap_couple_q_4 = CapNInterdigitalTee(design, 'Cap_Couple_Q_4',
                                    options=dict(
```

```
                                    pos_x='1mm',
                                    pos_y='0mm',
                                    orientation='180'
                                ))

gui.rebuild()
gui.autoscale()

cap_readout_q_1 = CapNInterdigital(design, 'Cap_Readout_Q_1',
                                options=dict(
                                    pos_x='-3mm',
                                    pos_y='4.8mm',
                                    orientation='0'
                                ))

cap_readout_q_2 = CapNInterdigital(design, 'Cap_Readout_Q_2',
                                options=dict(
                                    pos_x='3mm',
                                    pos_y='-4.8mm',
                                    orientation='180'
                                ))

cap_readout_q_3 = CapNInterdigital(design, 'Cap_Readout_Q_3',
                                options=dict(
                                    pos_x='-4.8mm',
                                    pos_y='-2mm',
                                    orientation='90'
                                ))

cap_readout_q_4 = CapNInterdigital(design, 'Cap_Readout_Q_4',
                                options=dict(
                                    pos_x='4.8mm',
                                    pos_y='2mm',
                                    orientation='-90'
                                ))

gui.rebuild()
gui.autoscale()

# Resonator Length Approximation from theory
from qiskit_metal.analyses.em.cpw_calculations import guided_wavelength
def find_resonator_length(frequency, line_width, line_gap, N):
    # frequency in GHz
    # line_width/line_gap in um
    # N -> 2 for lambda/2, 4 for lambda/4
    # substrate dimensions and properties already set (Default dielectric constant value is
    used)


    [lambdaG, effSqrt, q] = guided_wavelength(frequency*10**9, line_width*10**-6,
                                              line_gap*10**-6, 750*10**-6, 200*10**-9)

    return str(lambdaG/N*10**3) + " mm"

bus_13_length = find_resonator_length(5.8,10,2,2) #bus1
bus_23_length = find_resonator_length(6,10,2,2) #bus2
bus_24_length = find_resonator_length(6.2,10,2,2) #bus3
bus_14_length = find_resonator_length(6.4,10,2,2) #bus4

```

```
from collections import OrderedDict
jogs_start = OrderedDict()
jogs_start[0] = ['L', '350um']
jogs_start[1] = ['L', '500um']

jogs_end = OrderedDict()
jogs_end[0] = ['L', '680um']
jogs_end[1] = ['L','700um']

bus_1 = RouteMeander(design, 'Bus_1', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs = Dict(
                                                start_pin = Dict(
                                                    component = 'Q_1',
                                                    pin = 'bus_1'
                                                ),
                                                end_pin = Dict(
                                                    component = 'Cap_Couple_Q_1',
                                                    pin = 'second_end'
                                                )
                                            ),
                                            lead = Dict(
                                                start_straight = '125um',
                                                end_straight = '225um'
                                            ),
                                            meander = Dict(
                                                asymmetry = '50um'),
                                                fillet = "99um",
                                                total_length = '10.65mm'

                                        ))

jogs_start = OrderedDict()
jogs_start[0] = ['R', '350um']
jogs_start[1] = ['R', '700um']

bus_2 = RouteMeander(design, 'Bus_2', options = dict(
                                        hfss_wire_bonds = True,
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_2',
                                                pin = 'bus_2'
                                            ),
                                            end_pin = Dict(
                                                component = 'Cap_Couple_Q_2',
                                                pin = 'second_end'
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '25um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '10.3mm'

                                    ))

```

```
jogs_start = OrderedDict()
jogs_start[0] = ['R', '350um']
jogs_start[1] = ['R', '700um']

bus_3 = RouteMeander(design, 'Bus_3', options = dict(
                                        hfss_wire_bonds = True,
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_3',
                                                pin = 'bus_3'
                                            ),
                                            end_pin = Dict(
                                                component = 'Cap_Couple_Q_3',
                                                pin = 'second_end'
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '225um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '9.97mm'

                                    ))

jogs_start = OrderedDict()
jogs_start[0] = ['R', '350um']
jogs_start[1] = ['R', '700um']
bus_4 = RouteMeander(design, 'Bus_4', options = dict(
                                        hfss_wire_bonds = True,
                                        pin_inputs = Dict(
                                            start_pin = Dict(
                                                component = 'Q_4',
                                                pin = 'bus_4'
                                            ),
                                            end_pin = Dict(
                                                component = 'Cap_Couple_Q_4',
                                                pin = 'second_end'
                                            )
                                        ),
                                        lead = Dict(
                                            start_straight = '125um',
                                            end_straight = '225um'
                                        ),
                                        meander = Dict(
                                            asymmetry = '50um'),
                                            fillet = "99um",
                                            total_length = '9.65mm'

                                    ))

gui.rebuild()
gui.autoscale()

#Define Readout Resonators
Readout_1_length = find_resonator_length(6.8,10,6,2) #Readout_Q1
Readout_1_length = find_resonator_length(7,10,6,2) #Readout_Q2
```

```
Readout_1_length = find_resonator_length(7.2,10,6,2) #Readout_Q3
Readout_1_length = find_resonator_length(7.4,10,6,2) #Readout_Q4

jogs_start = OrderedDict()
jogs_start[0] = ['R', '350um']
jogs_end[0] = ['R', '300um']

readout_1 = RouteMeander(design, 'Readout_1', options=dict(
                                    hfss_wire_bonds=True,
                                    pin_inputs=Dict(
                                        start_pin=Dict(
                                            component='Q_1',
                                            pin='readout'
                                        ),
                                        end_pin=Dict(
                                            component='Cap_Readout_Q_1',
                                            pin='south_end'
                                        )
                                    ),
                                    lead=Dict(
                                        start_straight='125um',
                                        end_straight='125um',
                                        start_jogged_extension=jogs_start,
                                        end_jogged_extension=jogs_end
                                    ),
                                    meander=Dict(
                                        asymmetry='150um'),
                                        fillet='99um',
                                        total_length='8.95mm'

                                ))




readout_2 = RouteMeander(design, 'Readout_2', options=dict(
                                    hfss_wire_bonds=True,
                                    pin_inputs=Dict(
                                        start_pin=Dict(
                                            component='Q_2',
                                            pin='readout'
                                        ),
                                        end_pin=Dict(
                                            component='Cap_Readout_Q_2',
                                            pin='south_end'
                                        )
                                    ),
                                    lead=Dict(
                                        start_straight='125um',
                                        end_straight='125um',
                                        start_jogged_extension=jogs_start,
                                        # end_jogged_extension=jogs_end
                                    ),
                                    meander=Dict(
                                        asymmetry='150um'),
                                        fillet='99um',
                                        total_length='8.69mm'

```

```
                                    ))


readout_3 = RouteMeander(design, 'Readout_3', options=dict(
                                        hfss_wire_bonds=True,
                                        pin_inputs=Dict(
                                            start_pin=Dict(
                                                component='Q_3',
                                                pin='readout'
                                            ),
                                            end_pin=Dict(
                                                component='Cap_Readout_Q_3',
                                                pin='south_end'
                                            )
                                        ),
                                        lead=Dict(
                                            start_straight='125um',
                                            end_straight='125um',
                                            start_jogged_extension=jogs_start,
                                            # end_jogged_extension=jogs_end
                                        ),
                                        meander=Dict(
                                            asymmetry='150um'),
                                            fillet='99um',
                                            total_length='8.45mm'

                                    ))


readout_4 = RouteMeander(design, 'Readout_4', options=dict(
                                        hfss_wire_bonds=True,
                                        pin_inputs=Dict(
                                            start_pin=Dict(
                                                component='Q_4',
                                                pin='readout'
                                            ),
                                            end_pin=Dict(
                                                component='Cap_Readout_Q_4',
                                                pin='south_end'
                                            )
                                        ),
                                        lead=Dict(
                                            start_straight='125um',
                                            end_straight='125um',
                                            start_jogged_extension=jogs_start,
                                            # end_jogged_extension=jogs_end
                                        ),
                                        meander=Dict(
                                            asymmetry='150um'),
                                            fillet='99um',
                                            total_length='8.22mm'

                                    ))

jogs_start = OrderedDict()
jogs_start[0] = ['R', '250um']
jogs_end[0] = ['L', '200um']
```

```
512
513
514 charge_line_q_1 = RoutePathfinder(design, 'Charge_Line_Q_1', options = dict(
515                                         hfss_wire_bonds = True,
516                                         fillet='99um',
517                                         trace_width = '5um',
518                                         trace_gap = '3um',
519
520                                         lead=dict(
521                                             start_straight='125um',
522                                             end_straight='125um',
523                                             start_jogged_extension=jogs_start,
524                                             #end_jogged_extension=jogs_end
525                                             #end_straight='150um'
526                                             ),
527                                         pin_inputs=Dict(
528                                             start_pin=Dict(
529                                                 component='Launch_CL_Q_1',
530                                                 pin='tie'),
531                                             end_pin=Dict(
532                                                 component='Q_1',
533                                                 pin='Charge_Line')
534                                         )))
535
536
537
538
539
540 charge_line_q_2 = RoutePathfinder(design, 'Charge_Line_Q_2', options = dict(
541                                         hfss_wire_bonds = True,
542                                         fillet='99um',
543                                         trace_width = '5um',
544                                         trace_gap = '3um',
545
546                                         lead=dict(
547                                             start_straight='125um',
548                                             end_straight='125um',
549                                             start_jogged_extension=jogs_start,
550                                         ),
551                                         pin_inputs=Dict(
552                                             start_pin=Dict(
553                                                 component='Launch_CL_Q_2',
554                                                 pin='tie'),
555                                             end_pin=Dict(
556                                                 component='Q_2',
557                                                 pin='Charge_Line')
558                                         )))
559
560 flux_line_q_3 = RoutePathfinder(design, 'Flux_Line_Q_3', options = dict(
561                                         hfss_wire_bonds = True,
562                                         pin_inputs=Dict(
563                                             start_pin=Dict(
564                                                 component='Q_3',
565                                                 pin='flux_line'
566                                             ),
567                                             end_pin=Dict(
568                                                 component='Launch_FL_Q_3',
569                                                 pin='tie'
570                                             )
```

```
                                            ),
                                            fillet = '99um',
                                            trace_width = '5um',
                                            trace_gap = '3um'
                                        ))
flux_line_q_4 = RoutePathfinder(design, 'Flux_Line_Q_4', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
                                                start_pin=Dict(
                                                    component='Q_4',
                                                    pin='flux_line'
                                                ),
                                                end_pin=Dict(
                                                    component='Launch_FL_Q_4',
                                                    pin='tie'
                                                )
                                            ),
                                            fillet = '99um',
                                            trace_width = '5um',
                                            trace_gap = '3um'
                                        ))

gui.rebuild()
gui.autoscale()

tl_readout_q_1 = RouteStraight(design, 'TL_Readout_Q_1', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
                                                start_pin=Dict(
                                                    component='Cap_Readout_Q_1',
                                                    pin='north_end'
                                                ),
                                                end_pin=Dict(
                                                    component='Launch_Readout_Q_1',
                                                    pin='tie'
                                                )
                                            ),
                                            trace_width = '10um',
                                            trace_gap = '6um'
                                        ))
tl_readout_q_2 = RouteStraight(design, 'TL_Readout_Q_2', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
                                                start_pin=Dict(
                                                    component='Cap_Readout_Q_2',
                                                    pin='south_end'
                                                ),
                                                end_pin=Dict(
                                                    component='Launch_Readout_Q_2',
                                                    pin='tie'
                                                )
                                            ),
                                            trace_width = '10um',
                                            trace_gap = '6um'
                                        ))

tl_readout_q_3 = RouteStraight(design, 'TL_Readout_Q_3', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
```

```
                                                start_pin=Dict(
                                                    component='Cap_Readout_Q_3',
                                                    pin='north_end'
                                                ),
                                                end_pin=Dict(
                                                    component='Launch_Readout_Q_3',
                                                    pin='tie'
                                                )
                                            ),
                                            trace_width = '10um',
                                            trace_gap = '6um'
                                        ))

tl_readout_q_4 = RouteStraight(design, 'TL_Readout_Q_4', options = dict(
                                            hfss_wire_bonds = True,
                                            pin_inputs=Dict(
                                                start_pin=Dict(
                                                    component='Cap_Readout_Q_4',
                                                    pin='north_end'
                                                ),
                                                end_pin=Dict(
                                                    component='Launch_Readout_Q_4',
                                                    pin='tie'
                                                )
                                            ),
                                            trace_width = '10um',
                                            trace_gap = '6um'
                                        ))

gui.rebuild()
gui.autoscale()
gui.screenshot()
```

A.6 Application 1 of a 4 qubit chip

This code is written in Qiskit for demonstrating Entanglement Swapping using a 4-qubit chip.. . .

```

# A Basic Entanglement Swapping Circuit
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram

# Create a 4-qubit quantum circuit
qc = QuantumCircuit(4, 2)

# Step 1: Create entanglement (Bell pairs)
qc.h(0)
qc.cx(0, 1)
qc.h(2)
qc.cx(2, 3)

# Step 2: Bell-state measurement on qubits 1 & 2
qc.cx(1, 2)
qc.h(1)
qc.measure([1, 2], [0, 1])
```

```
20
21 # Step 3: Apply corrections based on measurement
22 qc.cx(0, 3)
23 qc.cz(1, 3)
24
25 # Visualize circuit
26 qc.draw('mpl')
```

A.7 Application 2 of a 4 qubit chip

This code is written in Qiskit for error detection using a 4-qubit chip.. . .

```
1  # Function to create a 4-qubit error detection circuit
2  def four_qubit_error_detection():
3      qc = QuantumCircuit(4, 2)  # 3 data qubits + 1 ancilla, 2 classical bits for measurement
4
5      # Encode logical |0> state with redundancy
6      qc.h(0)  # Hadamard on first qubit
7      qc.cx(0, 1)
8      qc.cx(0, 2)
9
10     # Introduce an artificial error (bit-flip on qubit 1 for testing)
11     qc.x(1)
12
13     # Syndrome measurement (parity checks)
14     qc.cx(0, 3)
15     qc.cx(1, 3)
16     qc.measure(3, 0)  # First parity check
17
18     qc.cx(2, 3)
19     qc.measure(3, 1)  # Second parity check
20
21     return qc
22
23 # Create the circuit
24 qc = four_qubit_error_detection()
25 qc.draw(output='mpl')
26 # Step 4: Run the Simulation
27 # Simulate the circuit
28 simulator = Aer.get_backend('qasm_simulator')
29 compiled_circuit = transpile(qc, simulator)
30 job = simulator.run(compiled_circuit)
31 result = job.result()
32
33 # Plot the result
34 plot_histogram(result.get_counts(qc))
```

Generated using Undegraduate Thesis LaTeX Template, Version 1.0. Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Tuesday 18th March, 2025 at 6:41pm.

B.Sc.

Engg.

EEE

BUET

4-Qubit Superconducting Quantum Chip

Fahim Shahriar Anim

March

2025