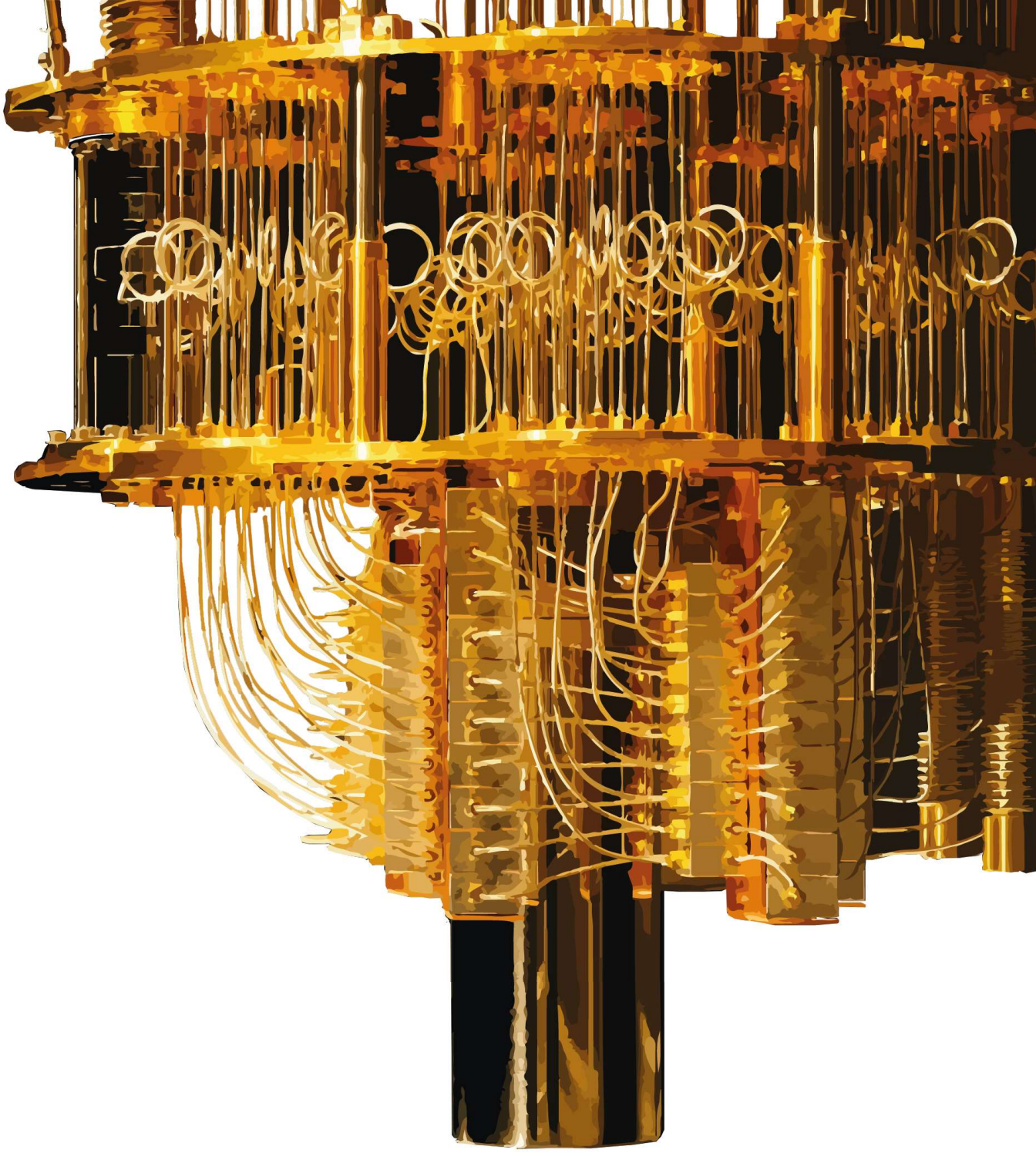


MASTER THESIS 2023 - 2024

Lucas Beyens
University of Antwerp

Randomized benchmarking of spin qubit systems

Theoretical approach and experimental implementation

Promotor | Prof. dr. ir. Kristiaan De Greve (IMEC)

Copromotor | Prof. dr. Jacques Tempere (University of Antwerp)

Supervisor | Dr. Clement Godfrin (IMEC)

The thesis is conducted under a non-disclosure agreement (NDA), between IMEC and the University of Antwerp.

imec

University of Antwerp
Faculty of Science

Abstract

Randomized Benchmarking (RB) is een efficiënte en robuuste methode die veel gebruikt wordt in experimentele set-ups om de gemiddelde nauwkeurigheid van een set logische poorten op een kwantumcomputer in te schatten. Het benchmarken van een kwantumcomputer is een schaalbaar proces dat robuust is tegen State-preparation-and-measurement (SPAM) errors. De scriptie bespreekt Standaard Randomized Benchmarking (SRB), de eenvoudigste methode om de gemiddelde nauwkeurigheid van een kwantumsysteem te bepalen, en Interleaved Randomized Benchmarking (IRB) om de kenmerken van een enkele logische poort op een kwantumcomputer te kunnen onderzoeken. Het voordeel van het gebruik van een RB-protocol is de mogelijkheid om SPAM-fouten te isoleren en tegelijkertijd zowel SPAM-fouten als specifieke fouten in de logische poorten te karakteriseren. Het nadeel is dat alleen de gemiddelde nauwkeurigheid van de logische poorten kan worden geëxtraheerd door de specifieke fouten te versterken in een depolariserend kanaal, terwijl de SPAM-fouten constant blijven. De fit parameters (A en B) geven dus de invloed van SPAM-fouten of de kwaliteit van de experimentele opstelling aan, wat een van de belangrijkste voordelen van RB is. De wiskundige inzichten om een RB-protocol af te leiden zijn drieledig: Ten eerste kan de gemiddelde nauwkeurigheid van de logische poorten worden bepaald als een kwantumkanaal zich gedraagt als een depolariserend kanaal. Ten tweede is een depolariserend kanaal een 'twirled' operatie van een unitary 2-design, zoals de Clifford-groep. Ten derde leiden speciefiek poort- en tijdonafhankelijke fouten tot een exponentiële afname van de nauwkeurigheid van het kwantumkanaal. Een fysische qubit wordt verkregen door een elektron in een kwantumdot te isoleren en een magnetisch veld toe te passen. De Zeeman-splitsing splitst de ontaarding van de energie van een geïsoleerd elektron in een $|0\rangle$- of $|1\rangle$-toestand, vergelijkbaar met de 0- of 1-waarden in een klassieke bit. De elektron-spin-resonantietechniek roteerd de qubit d.m.v. een radiofrequentiepuls. De qubit begint te roteren van de spin-up toestand naar de spin-down toestand en terug. De rotatieas kan verschoven worden door de fase in de radiofrequentiepuls te veranderen. De faseverschuiving kan experimenteel worden toegepast door IQ mixing waarbij een sinus- en cosinuspuls wordt gegenereerd via een AWG die een single-sideband up-conversion uitvoert. Hierdoor is de output een cosinus radiofrequentiepuls, waarbij de fase van de qubit kan worden veranderd door de fase van de cosinuspuls. Uiteindelijk genereert een Python Class sequenties om RB-experimenten uit te voeren op de experimentele opstelling bij Imec. Hierbij was het mogelijk om de willekeurige sequenties op de digitizer te genereren, wat aantoont dat de experimentele implementatie van het RB-protocol, zoals beschreven en uitgevoerd in deze scriptie, kan worden gebruikt voor verder onderzoek.

Abstract

Randomized Benchmarking is an efficient and robust method that is widely used in practice to estimate the average fidelity of a gate set implemented on a quantum computing device. Benchmarking a quantum computer is a scalable process that is robust to state-preperation-and-measurement (SPAM) errors. The thesis discusses Standard Randomized Benchmarking (SRB), the most hands-on method to extract the average fidelity of a quantum system, and Interleaved Randomized Benchmarking (IRB) to retrieve the characteristics of a single gate on a quantum device. The advantage of using an RB protocol is the ability to isolate SPAM errors and simultaneously characterize both SPAM errors and gate errors. A drawback is that only the average gate fidelity can be extracted by amplifying the gate errors in a depolarizing channel while leaving the SPAM errors constant. Hence, the A and B parameters indicate the influence of SPAM errors or the quality of the experimental setup, which is one of the major advantages of RB. The mathematical insights to derive an RB protocol are threefold: First, the average gate fidelity can be extracted if a quantum channel behaves as a depolarizing channel. Second, a depolarizing channel is a twirled operation of a unitary 2-design, such as the Clifford group. Third, gate and time-independent errors lead to an exponential fidelity decay. A physical qubit is obtained by isolating an electron in a quantum dot and applying a magnetic field. The Zeeman splitting splits the degeneracy of the energy of an isolated electron in a $|0\rangle$- or $|1\rangle$-state. Comparable with the 0 or 1 values in a classical bit. The electron spin resonance technique drives the current through a radiofrequency pulse. The qubit starts to rotate from the spin-up state to the spin-down state and back. The rotation axis can be shifted by changing the phase in the radiofrequency pulse. The phase shift can be experimentally applied by IQ mixing where a sine and cosine wave are generated through an AWG performing single-sideband upconversion so the output is a cosine radiofrequency pulse. The phase of the qubit can be regulated by changing the phase of the cosine wave. Eventually, a Python Class generates sequences to execute RB experiments on the experimental set-up at Imec. The thesis's outcome shows the ability to generate random sequences on the digitizer, proving that the RB protocol's experimental implementation, as described and executed in this thesis, can be used for further research purposes.

Acknowledgments

First, I would like to express my gratitude to Professor Kristiaan De Greve for allowing me to conduct this research. I had the privilege of working at Imec for one year, an experience for which I am incredibly thankful. Working at Imec as a student was a fantastic opportunity, and it has been immensely valuable for my professional development. Additionally, I wish to thank Dr. Clement Godfrin, who is always cheerful and never hesitates to answer any questions, for his guidance throughout this process. I have learned an extraordinary amount from him and will cherish the memories of our meetings. Clement, c'était super, merci beaucoup.

Further, I would like to express my appreciation to all the professors and assistants at the University of Antwerp. Their collective efforts and dedication have made the past five years of my academic journey challenging yet fulfilling, and I am proud to have been a part of this academic community. Finally, I would also like to thank my parents, sister, grandparents, and friends for their unconditional support during the study period. Results are always the merits of the people around you who make it possible. My sincere thanks for that.

Lucas Beyens

Contents

1 Introduction **1**
1.1 Thesis outline . . . 3

2 Mathematical background of applied quantum mechanics **5**
2.1 From bits to qubits . . . 5
2.2 Quantum gates . . . 8
2.3 Overview of quantum channels and measurement fidelity . . . 10
2.3.1 Quantum channels . . . 10
2.3.2 Deplorazing channels . . . 12
2.3.3 The Pauli transfer matrix and the superoperator formalism . 13
2.3.4 Fidelity and Measurement of quantum channels . . . 15

3 Theoretical approach of the Standard and Interleaved Randomized Benchmarking protocol **19**
3.1 The exact Haar Twirl . . . 19
3.2 Unitary 2-designs . . . 22
3.3 The one qubit Clifford group $\mathcal{C}_1$ . . . 23
3.4 The standard Randomized Benchmarking protocol . . . 24
3.4.1 Expected fidelity decay for random Clifford sequences . . . 24
3.4.2 The standard Randomized Benchmarking protocol . . . 28
3.5 Interleaved Randomized Benchmarking . . . 29

4 Simulations of the Standard and Interleaved Randomized Benchmarking experiments **31**
4.1 Code outline . . . 31
4.2 Simulations of standard RB experiments . . . 34
4.2.1 The AER noise models . . . 34
4.3 Simulations of interleaved RB experiments . . . 40
4.4 Simulations: Fidelity decays . . . 42

5 Practical implementation of a Randomized Benchmarking protocol **45**
5.1 The quantum dot . . . 45
5.2 A physical qubit . . . 48

5.3 Electron spin resonance: Applying quantum gates 51
5.4 Calibration 58
5.5 Practical implementation of the RB protocol 59
5.6 Experimental set-up 63
5.6.1 IQ-mixer 65

6 Conclusion **71**
6.1 outlook 72

A Common quantum gates **73**
A.1 Single qubit Clifford gates 73
A.2 Single qubit rotation gates 75
A.3 Unitary gates U 76

B Ladder operators $\hat{\sigma}_+$ and $\hat{\sigma}_-$ **77**
B.1 single qubit ladder operators 77

C Generated data **79**
C.1 Simulation: Aer noise model 79
C.2 experimental implementation: RB results 82
C.3 Generated sequences 82

D Python Code **85**
D.1 Simulations: Noise model 85
D.2 Simulations: ONEQ_RBClass 86
D.3 Class ONEQ_SRB_experiment 94

List of Abbreviations

RB Randomized Benchmarking

SRB Standard Randomized Benchmarking

IRB Interleaved Randomized Benchmarking

SPAM State preparation and measurement errors

CPTP Complete positive and trace preserving

POVM Positive operator value measurement

PTM Pauli transfer matrix

RF radiofrequency

SiQD's Silicon quantum dots

SET Single electron transistor

ESR Electron spin resonance

REW Rotating wave approximation

FWHM Full width at half maximum

AWG Arbitrary wave generator

PSG Performance signal generator

IQ In-phase and quadrature

LO Local oscillator

Chapter 1

Introduction

Quantum computers are considered one of the most promising technologies of this century, due to their ability to harness the principles of quantum mechanics to perform calculations at unprecedented speeds. Quantum parallelism enables quantum computers to process vast amounts of information in parallel, potentially solving complex problems that are intractable for classical computers. Quantum computing holds the promise of revolutionizing fields such as cryptography, drug discovery, optimization, and machine learning [1, 2].
The thesis focuses on silicon spin qubit systems used at Imec to perform quantum computing. Silicon spin qubit systems are a promising approach to the search for scalable and fault-tolerant quantum computing architectures [3]. Systems based on silicon semiconductor technology harness the spin states of individual electrons or nuclei confined within silicon-based nanostructures as quantum bits (qubits). In this thesis, only individual electrons are used to build up a physical qubit. Silicon's exceptional material properties, including isotopic purity resulting in relatively long coherence times, make it an attractive platform for realizing stable and reproducible qubits [4, 5, 6, 7]. The foundation of silicon spin qubits lies in manipulating and controlling electron spins, typically achieved through applying magnetic fields, using the Zeeman effect. Quantum information is encoded in the spin states of these particles, with coherent driving performed using microwave or radiofrequency pulses. Silicon spin qubits benefit from decades of research and development in silicon-based electronics, enabling compatibility with existing fabrication techniques and classic circuit control integration.
One of the major advantages of silicon spin qubits is their capability to perform single- and multi-qubit operations, i.e., the potential for scalability. Together with their feasibility of quantum error correction protocols [8, 9, 4].

As more and larger quantum computers arise, there is a growing need for methods to extract the fidelity of the outcomes and specific gate characteristics. Straightforward tomography-based methods, such as Quantum Process Tomography (QPT), provide information about a single gate and require measurements that scale ex-

ponentially with the number of qubits. Therefore, it is not scalable. In addition, tomography-based methods depend highly on state-preparation and measurement errors (SPAM) errors [10, 11]. To overcome these liabilities, a new approach, such as Randomized Benchmarking (RB), is an efficient and robust method and is widely used in practice for estimating the average fidelity of a gate set implemented on a quantum computing device and was first introduced by E. Knill in 2008 [12]. Benchmarking a quantum computer is a scalable process, robust to SPAM errors [10, 13]. The thesis discusses Standard Randomized Benchmarking (SRB), the most hands-on method to extract the average fidelity of a quantum system, and Interleaved Randomized Benchmarking (IRB) to retrieve the characteristics of a single gate on a quantum device.

SRB serves as the foundational form of randomized benchmarking, originally introduced as a figure of merit to quantify the average error rate of quantum gates without requiring detailed knowledge of the underlying noise sources. The core principle of SRB involves repeatedly applying a sequence of randomly chosen gate operations to a quantum system, followed by a measurement to determine the fidelity of the gate set. By analyzing the sequence fidelity decay with increasing sequence length, SRB provides valuable information about the average gate error rate, offering a comprehensive approach to gate performance over a wide range of experimental conditions [14, 15]. A drawback is the gate independence of this protocol, for example, SRB may overlook specific error mechanisms that are not captured by the randomized gate sequences.
In addition, IRB incorporates interleaved gates alongside the target gates to probe specific error mechanisms. By systematically varying the interleaved gates and analyzing their impact on the overall fidelity, IRB provides deeper insights into the nature and origins of errors affecting quantum gates, enabling more refined error mitigation strategies through analyzing the ratio of the SRB and IRB decay parameters [16, 17, 7]. While SRB offers a straightforward and efficient means to estimate average gate error rates, IRB provides a more nuanced understanding of gate performance by isolating and characterizing individual error contributions. Both protocols will be mathematically derived, discussed, and simulated in the thesis using the IBM quantum environment [18].

Gates are physically generated using the electron spin resonance (ESR) technique, where radiofrequency pulses drive the qubit around a certain axis to induce π- and $\pi/2$-pulses. A phase difference in the pulse causes a qubit rotation in the (x, y)-plane, making it feasible to physically implement quantum computational gates. Hence, a pulse with a series of phase differences physically defines a gate sequence. The thesis will cover the experimental implementation of these pulses for the one-qubit spin quantum device at Imec.

1.1 Thesis outline

Overall, the thesis consists of three major parts: The mathematical approach to RB, simulating SRB and IRB experiments, and the practical implementation of RB sequences on the quantum device at Imec.

- **Chapter 2: Mathematical background of applied quantum mechanics**
  Chapter 2 provides some background information about the difference between classical bits and qubits, continuing with the difference between classic logic gates and quantum logic gates. In addition, a broad mathematical background in quantum channels and fidelity is needed to understand the derivations in Chapter 3.

- **Chapter 3: Theoretical approach of the Standard and Interleaved Randomized Benchmarking protocol**
  Chapter 3 provides a theoretical approach to the RB protocol, where some fundamental mathematical topics are discussed, such as twirling quantum channels, Clifford operators, and fidelity decay in randomized sequences. At the end, a distinction is made between SRB and IRB.

- **Chapter 4: Simulations of Standard and Interleaved Randomized Benchmarking experiments**
  Chapter 4 provides the simulations of both one-qubit SRB and IRB experiments using the IBM quantum environment and Qiskit, an open-source Python package from IBM. Simulations are performed exclusively on local devices.

- **Chapter 5: Practical implementation of gate sequences on a silicon spin qubit system**
  The last chapter provides a practical implementation of quantum gates on a silicon spin qubit system, using the ESR technique. First, a mathematical description of the ESR technique is essential to link the coherent driving pulse of a qubit with a quantum gate. Next, we will look into the experimental setup and how an RB experiment can be performed on it.

- **Chapter 6: Conclusion and outlook**
  At last, the thesis is summarized in Chapter 6 and raises a few questions for further research in the outlook.

Chapter 2

Mathematical background of applied quantum mechanics

The chapter first introduces the concept of qubits and the evolution of quantum mechanical systems. Following by the concept of quantum gates is discussed in more detail, and some mathematical techniques are eventually required to understand the derivations in the next chapter. These techniques contain the concept of Kraus operators, depolarizing channels, advanced quantum state representations, and the concept of quantum state fidelity.

2.1 From bits to qubits

In a classical computer, a bit (representation of a binary digit) can hold 0 or 1. On-off switches in the hardware generate bits, forming patterns that represent other numbers and, in the case of computers, letters, and symbols. Transistors are commonly used as on-off switches in classic computer hardware. Moore's law expected that the number of transistors in an integrated circuit doubles every two years [19]. These days, the increase in computational power of classical computers is flattened. The transistors are getting too small causing malfunctions due to quantum mechanical effects, such as quantum tunneling. However, quantum mechanics may provide a solution in terms of qubits and, in general, quantum computers.

A qubit or quantum bit is the counterpart of the conventional binary digit (bit) and the fundamental unit of quantum information in quantum computing and quantum information systems. A qubit can exist in a superposition of both (0, 1)-states, making it a more powerful tool than the conventional binary bit [4, 20]. The quantum state of a qubit is given by

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

where $|\alpha|^2 + |\beta|^2 = 1$ and $\alpha, \beta \in \mathbb{C}$. This is the simplest form for a qubit representation. A more intuitive notation uses the Bloch sphere to depict the qubit state and can be written as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{2.2}$$

where θ is the polar angle and ϕ the azimuthal. A graphical representation is given in figure 2.1.

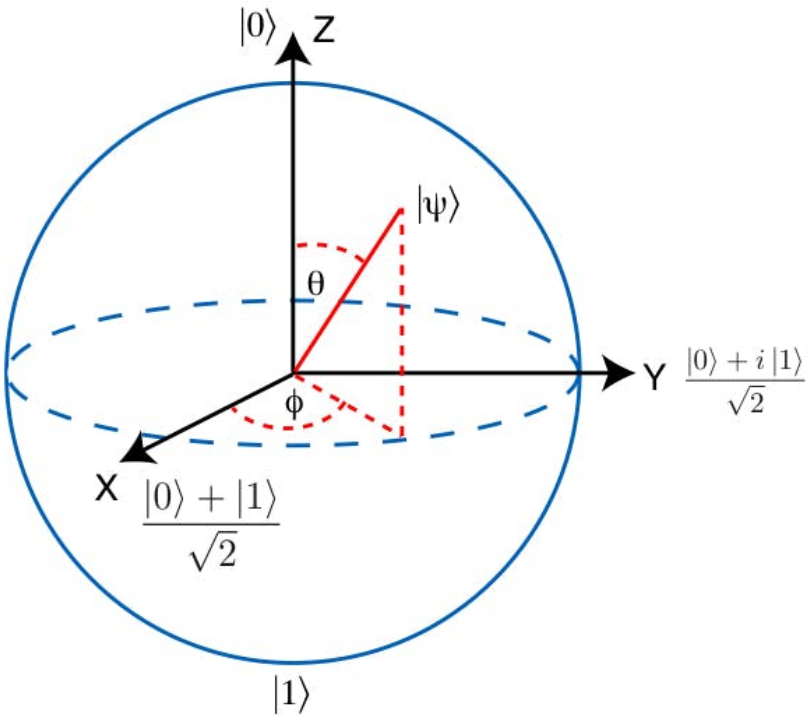


Figure 2.1: Graphical representation of the Bloch sphere. When the qubit state $|\Psi\rangle$ is located in the equatorial plane, the $|0\rangle$-and-$|1\rangle$-states are both equally populated. Which results in maximal superposition of the quantum state. Source: Quantum Inspire (QuTech)

The changes of a quantum state over time can be described by the Schrödinger equation. The time-dependent Schrödinger equation states:

$$i\hbar\frac{\partial}{\partial t}|\Psi t\rangle = H|\Psi(t)\rangle\,, \tag{2.3}$$

and is the most general form of the Schrödinger equation, where $\hbar$ is the reduced Planck constant and H the Hamiltonian operator. The wavefunction $|\Psi(t)\rangle$

represents the probability amplitude of the quantum state and $i\hbar\frac{\partial}{\partial t}$ the energy operator. The evolution of the quantum state $|\Psi(t)\rangle$ from time t_0 to t, can be described through the time-evolution operator (or propagator) $U(t, t_0)$ as

$$|\Psi(t)\rangle = U(t, t_0)\,|\Psi(t_0)\rangle\,, \tag{2.4}$$

and $|\Psi(t_0)\rangle$ the initial quantum state and $U(t, t_0)$ a unitary propagator. For a time-dependent Hamiltonian, the unitary propagator is given by

$$U = \exp\left\{-i\int H(t)\,dt\right\}, \tag{2.5}$$

if $[H(t), H(t^{'})] = 0$ for any combination of t and $t^{'}$ [4].
Another commonly used representation is through density matrices, where the off-diagonal elements depict the mixed states of a qubit.

$$\rho = \sum_j p_j\,|\psi_j\rangle\,\langle\psi_j|\,, \tag{2.6}$$

where $\rho \in \mathcal{H}_d$ and $d = 2^n$ with n the number of qubits. $\mathcal{H}_d$ is the d-dimensional Hilbert space. The evolution of the quantum state is therefore given by

$$\mathcal{U}(\rho_0) = \rho(t)U(t, t_0)\rho(t_0)U^{\dagger}(t, t_0). \tag{2.7}$$

A qubit must meet certain criteria to build a gate-based quantum computer. DiVincenzo's criteria, proposed by theoretical physicist David P. Di Vincenzo in 2000, explain the physical implementation requirements for quantum computers. The DiVincenzo criteria described in [4] are listed below:

1. **Scalability**: The qubit system should be scalable to a large number of qubits, allowing for the construction of a quantum computer with a sufficient number of qubits to perform meaningful computations. In addition, a qubit must be "well-characterized", meaning the internal Hamiltonian of the qubit, couplings with other states, or interactions with other qubits are well known. As well as the couplings to external fields to drive the qubit. Ignorance of these matters leads to decoherence of the qubit.

2. **Qubit Initialization**: The ability to initialize the qubits to a well-defined state. If the purity of the initialized state is too low, the use of the qubit for quantum computing results in incorrect outcomes. In addition, the measurement qubits should be initialized before every manipulation. A typical read-out sequence consists of three steps: initialization, manipulation, and read-out. Hence, the initialization needs to be fast enough.

3. **Long Qubit Coherence Time**: Qubits should have a sufficiently long coherence time [1], meaning they can maintain their quantum state for a duration long enough to perform quantum computations. The coherence time or dephasing time is the time for the quantum state to be lost by $1/e$ without any correction and is denoted by T_2^*. To perform decent calculations, The decoherence time needs to be much longer than the time needed to perform a π-pulse.

4. **Universal Set of Quantum Gates**: The ability to perform a universal set of quantum logic operations (quantum gates) on the qubits, allowing for the implementation of any quantum algorithm. Hence, any unitary operator should be able to be implemented as a quantum gate.

5. **Qubit Measurement**: The ability to measure the qubits accurately and reliably, extracting the results of quantum computations. The read-out fidelity determines the reliability in the absence of other imperfections, the reliability is also sometimes referred to as quantum efficiency.

2.2 Quantum gates

Assume a problem where a yes-or-no (boolean) question needs to be solved for each asset. For ten assets, the problem space becomes $2^{10} = 1024$. It is doable for almost all classic algorithms on our personal computers. But if we expand our collection from ten to a hundred assets, the problem space becomes $2^{100} = 1\ 267\ 650\ 600\ 228\ 229\ 401\ 496\ 703\ 205\ 376$ a problem that can no longer be solved with a personal computer and even very difficult (and maybe impossible) for the world's most powerful supercomputers.
Due to the enormous size of the problem space, computer scientists these days are heading more toward quantum computing, where qubits are used instead of classical bits. Working with qubits gives three major advantages [21]:

1. **Superposition**: While a classical bit can only be in the state 0 or 1, a qubit can have the state 0, 1, or a superposition of 0 and 1 (and thus an infinite amount of states).

[1] The T_2^* time, often referred to as the "dephasing time" or "coherence time," is a measure of how long a qubit can maintain its superposition state before losing coherence due to environmental noise. In other words, it indicates the timescale over which the quantum information encoded in the qubit's state remains intact [21, 4]. The T_2 time refers to the natural dephasing time caused by atomic or molecular interactions [22]. Whereas the T_2^* time refers to the observed dephasing time caused by other secondary factors such as magnetic fields, inhomogeneities, etc. In addition, the T_1 is used to indicate the relaxation time or amplitude damping and indicates the loss of energy from the system. [23]

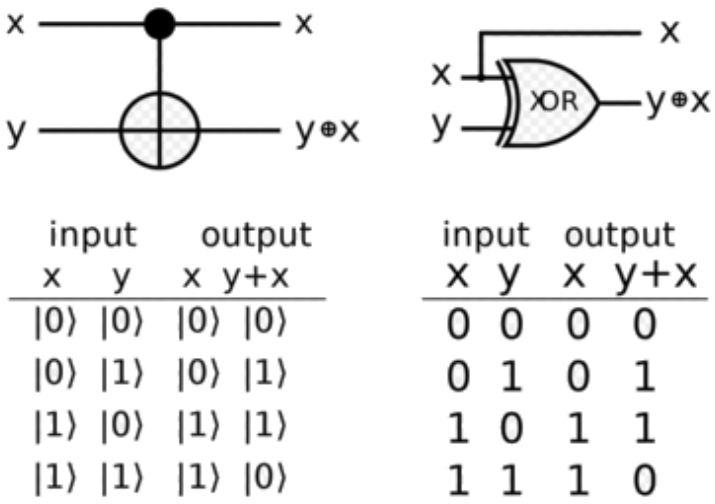
2. **Entanglement**: Entanglement is a phenomenon where the quantum states of two or more qubits become correlated in such a way that the state of one qubit depends on the state of the others, even when they are physically separated. This correlation enables the qubits to share information instantaneously, regardless of the distance between them.

3. **Interference**: Quantum interference is a phenomenon where the probability amplitudes of different quantum states interfere constructively or destructively, depending on their relative phases. This interference allows quantum computers to amplify the probability of obtaining the correct answer while suppressing the probability of obtaining incorrect answers. Quantum algorithms exploit interference to enhance computational efficiency and speed.

These three characteristics of quantum parallelism are fundamental features of quantum computing that underpin its potential for achieving exponential speedups in certain computational tasks. Using quantum parallelism effectively requires an intensive and precise development of quantum algorithms and techniques to exploit the unique properties of quantum systems. The mechanisms used in computers to think and execute commands are logic gates, they have a quantum equivalent called quantum logic gates. Although both gates of the same family will give the same answer, due to quantum parallelism, the underlying behavior of quantum logic gates is completely different from the classical ones. Figure 2.2 depicts the quantum CNOT gate and the classical XOR gate, and as seen in the figure, the outcome of both gates is identical.

Quantum gates exploit quantum parallelism to perform computations on multiple states simultaneously. Classical logic gates are not inherently reversible, while quantum gates are reversible. This property of quantum gates is essential for preserving quantum coherence and enabling quantum algorithms to run efficiently. Overall, while classical and quantum logic gates serve similar functions in their respective computing paradigms, their underlying principles, behaviors, and capabilities are fundamentally different due to the unique properties of classical and quantum systems.

Note that a quantum gate is mathematically equal to unitary operations. The simplest form of writing a quantum state is via a quantum statevector. The final state of a quantum system can be calculated by applying the unitary operations, represented by the gates, on the statevector representing the initial state. The outcome is again a statevector. Applying unitary operators, which is the same as executing a quantum circuit, requires $2^n \times 2^n$ matrices, with n the number of qubits. The next section gives a mathematical overview of quantum channels and how to work with them to obtain the fidelity.



input		output	
x	y	x	y+x
|0⟩	|0⟩	|0⟩	|0⟩
|0⟩	|1⟩	|0⟩	|1⟩
|1⟩	|0⟩	|1⟩	|1⟩
|1⟩	|1⟩	|1⟩	|0⟩

input		output	
x	y	x	y+x
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Figure 2.2: Comparison of classic and quantum logic gates.
Left: The quantum logic CNOT gate. Right: The classical logic XOR gate. The outcome of both gates is identical, although the behavior is different.
Source: Einstein Relatively easy [24]

2.3 Overview of quantum channels and measurement fidelity

2.3.1 Quantum channels

Quantum channels can be represented as a series of gates, i.e., unitary operations, transformations, and measurements [21, 25, 26]. In real devices, gates contain noise and will not return the pure statevector. Consider a pure quantum state ρ, the transformation of the quantum state in a quantum channel is given by

$$\mathcal{G} : \rho \longrightarrow \mathcal{G}(\rho) \in \mathcal{H}_d, \tag{2.8}$$

and can be seen as a linear operation on density matrices. The map $\mathcal{G}$ represents the quantum channel and $\mathcal{G}(\rho)$ the final state after the process occurs. A quantum system can be divided into two subsystems: the system of interest or principal system and the environment, forming a closed quantum system together. Assume the system-environment state as a product state $\rho \otimes \rho_{env}$, the final state of ρ after some unitary transformation U can be extracted by taking the partial trace over the environment:

$$\mathcal{G}(\rho) = \mathrm{Tr}_{env}\left[\big(U(\rho) \otimes \rho_{env}\big)U^{\dagger}\right]. \tag{2.9}$$

The final state $\mathcal{G}(\rho)$ can be seen as the reduced state of the system. Another way of describing quantum channel is through the Operator-sum representation:

$$\mathcal{G}(\rho) = \sum_i \langle e_i| U \big[\rho \otimes |e_0\rangle \langle e_0|\big] U^\dagger |e_i\rangle \tag{2.10}$$

$$= \sum_i K_i \rho K_i^\dagger, \tag{2.11}$$

where $K_i = \langle e_i| U |e_0\rangle$ and completeness relation

$$\sum_i K_i^\dagger K_i = 1. \tag{2.12}$$

Equation 2.10 is known as the Kraus representation. The Kraus operators K_i are Completely Positive Trace Preserving CPTP maps, which means that the final state after application of the map has nonnegative probabilities for measuring the eigenstate of a certain observable (i.e., the density matrix is always semidefinite). CPTP map on a subsystem ensures a valid final quantum state. Trace-preserving means there is no leakage in the channel or "conservation of probability":

$$1 = \sum_m p(m) = \mathrm{Tr}\Big[\big(\sum_m \mathcal{G}_m\big)(p)\Big]. \tag{2.13}$$

A non-trace-preserving map (or quantum channel) results in

$$\sum_k K_i^\dagger K_i < 1 \longrightarrow \mathrm{Tr}\big(\mathcal{G}(\rho)\big) < \mathrm{Tr}(\rho). \tag{2.14}$$

Kraus operators are commonly used to describe quantum channels and quantum error correction, and according to [21], if and only if a map has an operator-sum representation it satisfies these axioms:

1. A1: $Tr[\mathcal{G}(\rho)]$ is the probability the process represented by $\mathcal{G}$ occurs, when ρ is the initial state. Thus, $0 \leq Tr[\mathcal{G}(\rho)] \leq 1$ for any state ρ.

2. A2: $\mathcal{G}$ is a *convex-linear map* on the set of density matrices, that is, for probabilities p_i: $\mathcal{G}\big(\sum_i p_i \rho_i\big) = \sum_i p_i \mathcal{G}(\rho_i)$.

3. A3: $\mathcal{G}$ is a *completely positive* map. That is, if $\mathcal{G}$ maps density operators of system Q_1 to density operators of system Q_2, then $\mathcal{G}(A)$ must be positive for any positive operator A.

The above axioms are just to indicate that Kraus operators are well-defined operators, and we do not need to worry about some mathematical nuances or difficulties. However, composing channels becomes ugly very quickly

$$\mathcal{A} \circ \mathcal{B} = \sum_{i=1}^{M} \sum_{j=1}^{N} A_i B_i \rho B_i^\dagger A_i^\dagger, \tag{2.15}$$

and is a drawback of using Kraus operators. Subsequently, the Kraus representation of quantum channels is not unique.

2.3.2 Deplorazing channels

A valuable quantum channel is a depolarizing channel, describing a noisy channel where quantum information dissipates equally in all directions [21, 7, 11]. The most important application of the depolarizing channel is the randomized benchmarking protocol, which is the scope of this thesis. Depolarizing noise serves as a powerful noise model for analyzing the performance of error correction circuits. For a single qubit, a maximally depolarizing state can be achieved by the operator:

$$\mathcal{K}_{dep_{max}}(\rho) = \frac{\rho + X\rho X + Y\rho Y + Z\rho Z}{4} = \frac{I}{2}. \tag{2.16}$$

For any ρ, will the depolarizing channel return a completely mixed state, i.e., a depolarized state. If the quantum channel is depicted as an n-qubit CPTP map using Kraus operators, $\Lambda(\rho) = \sum_{i=1}^{N} K_i \rho K_i^\dagger$, the maximum-depolarizing channel gives rise to the Kraus operators in the form: $I/2$, $X/2$, $Y/2$ and $Z/2$.
The depolarizing channel can be generalized as:

$$\mathcal{K}_{dep} = (1-p)\rho + p\frac{I}{d} \tag{2.17}$$

$$= \frac{1-3p}{4}\rho + \frac{p}{4}(X\rho X + Y\rho Y + Z\rho Z), \tag{2.18}$$

where the quantum state ρ is preserved with probability $p-1$ and with probability p the state is depolarized. The parameter p is therefore known as the depolarizing parameter, and can be seen as a measurement for the infidelity of the quantum channel. The dimension $d = 2^n$, for n the number of qubits. The Kraus operators in this form are given through $\sqrt{1-3p/4}I$, $\sqrt{p}X/2$, $\sqrt{p}Y/2$ and $\sqrt{p}Z/2$.
When $\mathcal{K}_{dep}$ is repeated N times

$$\lim_{N\to\infty} \mathcal{K}_{dep}^N(\rho) = \mathcal{K}_{dep_{max}} = I, \tag{2.19}$$

the quantum state evolves in a totally mixed state, a characteristic of the depolarizing channel which allows us to experimentally extract the depolarizing error rate [7]. The depolarizing channel will later prove useful in the derivation of the randomized benchmarking protocol.

2.3.3 The Pauli transfer matrix and the superoperator formalism

The Kraus operators can be further decomposed into the Pauli basis $K_i = \sum_{j=1}^{d^2} a_{ij} P_j$ and a $\mathcal{X}$-matrix which leads to a useful representation of a quantum map [11, 7]:

$$\begin{aligned}\Lambda(\rho) &= \sum_i K_i \rho K_i^\dagger = \sum_{jk} \big(\sum_i a_{ij} a_{ik}^* \big) P_j \rho P_k \\ &= \sum_{j,k=1}^{N} \mathcal{X}_{jk} P_j \rho P_k,\end{aligned}$$

where $N = d^2$ and $\mathcal{X}_{ij} = \sum_i a_{ij} a_{ik}$, a $d^2 \times d^2$ complex-valued matrix, Hermitian and positive semidefinite, hence, holds the completeness relation $\sum_{jk} \mathcal{X}_{jk} P_k P_j = I$. The $\mathcal{X}$-matrix completely determines the map Λ. The Pauli operators P_i belongs to the n-qubit Pauli group $\{I, X, Y, Z\}^{\otimes n} = \mathcal{P}^{\otimes n}$.

Using the Kraus operators from equation 2.17, the $\mathcal{X}$-matrix for the depolarizing channel can be written as

$$\mathcal{X}_{dep} = \begin{pmatrix} \frac{1-3p}{4} & 0 & 0 & 0 \\ 0 & \frac{p}{4} & 0 & 0 \\ 0 & 0 & \frac{p}{4} & 0 \\ 0 & 0 & 0 & \frac{p}{4} \end{pmatrix}. \tag{2.20}$$

While the process matrix experiences significant attention in the literature, it's important to note that the $\mathcal{X}$-matrices are not suited for direct multiplication. Therefore, determining the $\mathcal{X}$-matrix of a quantum circuit (or channel) doesn't involve simply multiplying the $\mathcal{X}$-matrices of individual quantum operations within the channel. Superoperators, on the other hand, are designed to overcome this issue and conveniently describe quantum operations in channels. The Pauli Transfer Matrix (PTM) is a widely used representation of superoperators and can be written as:

$$\Lambda_{ij}^{\mathcal{K}} = \frac{1}{d} \operatorname{Tr}\left[P_i \mathcal{K}(P_j)\right], \tag{2.21}$$

maps a Pauli operator P_j (input) to another Pauli operator P_i (output) with coefficients $\Lambda_{ij}^{\mathcal{K}}$ [7]. The PTM for a depolarizing channel is given by

$$\Lambda_{dep} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-p & 0 & 0 \\ 0 & 0 & 1-p & 0 \\ 0 & 0 & 0 & 1-p \end{pmatrix}. \tag{2.22}$$

In general, the superoperator formalism states that density operators $\rho \in \mathcal{H}_d$ are represented as vectors $|\rho\rangle\rangle$ in the Hilbert-Schmidt space of dimension d^2 [11, 27]. Linear maps on density operators, i.e. quantum operations, are represented as matrices of dimension $d^2 \times d^2$. The Hilbert-Schmidt inner product is defined as

$$\langle\langle A|B\rangle\rangle = \frac{\mathrm{Tr}\{A^\dagger B\}}{d}, \tag{2.23}$$

where A and B are density operators. The map composition is represented as matrix multiplication through the definition of the inner product, a diagonalization can be performed to easily extract the trace of the matrix multiplication. It is convenient to write the superoperators in the Pauli basis, using normalized Pauli operators $P_i \to P_i/\sqrt{d} = \{I/\sqrt{d}, X/\sqrt{d}, Y/\sqrt{d}, Z/\sqrt{d}\}$, the superoperator therefore given by

$$|\rho\rangle\rangle = \frac{1}{d}\sum_{i=1}^{d^2} |P_i\rangle\rangle \langle\langle P_i| |\rho\rangle\rangle, \tag{2.24}$$

where $\langle P_i|\rho\rangle = \mathrm{Tr}\{P_i\rho\}$. Note that superoperators can be defined to any basis of $\mathcal{H}_n$. As an example, the density matrix of a single qubit can be expanded in the Pauli basis:

$$\rho = \frac{1}{2}(I + r_x X + r_y Y + r_z Z), \tag{2.25}$$

and $\boldsymbol{r} = (r_x, r_y, r_z)$ the Bloch vector. In the superoperator formalism, a single qubit state is represented by

$$|\rho\rangle\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathrm{Tr}(\rho I) \\ \mathrm{Tr}(\rho X) \\ \mathrm{Tr}(\rho Y) \\ \mathrm{Tr}(\rho Z) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ r_x \\ r_y \\ r_z \end{pmatrix}. \tag{2.26}$$

2.3.4 Fidelity and Measurement of quantum channels

The fidelity is a useful metric that represents the distance between quantum states [21, 26]. The fidelity between state ρ and $\rho^{'}$ can be written as

$$F(\rho, \rho') = \mathrm{Tr}\left[\sqrt{\sqrt{\rho}\rho'\sqrt{\rho}}\right] \in [0, 1], \tag{2.27}$$

and is a symmetric comparison between the two states. If the outcome is 1, the states are identical. If $F(\rho, \rho') < 1$, then $\rho \neq \rho'$ and the corresponding purifications or eigenvectors: $|\psi\rangle \neq |\varphi\rangle$. When calculating the fidelity between a state ρ and a pure state $|\psi\rangle$, it can be seen that

$$F(|\psi\rangle, \rho) = \mathrm{Tr}\left[\sqrt{\langle\psi|\rho|\psi\rangle\,|\psi\rangle\langle\psi|}\right], \tag{2.28}$$

the fidelity is equal to the square root of the overlap between the state ρ and the pure state $|\psi\rangle$. Another property of fidelity is its invariance under unitary transformations [21]:

$$F(U\rho U^{\dagger}, U\rho' U^{\dagger}) = F(\rho, \rho'). \tag{2.29}$$

The fidelity can be written in multiple forms, using the $\mathcal{X}$-matrices and the Hilbert-Schmidt inner product from equation 2.20 and 2.23, the process or entanglement fidelity [7] can be written as

$$F_p = \mathrm{Tr}[\mathcal{X}_{exp}\mathcal{X}_{ideal}] \tag{2.30}$$

$$= \mathrm{Tr}\left[\Lambda_{exp}\Lambda_{ideal}^{-1}\right], \tag{2.31}$$

where the second expression is the same using PTM representation. A general expression for the average gate fidelity of the noisy experimental implementation $\tilde{\mathcal{G}}$ of an ideal unitary channel $\mathcal{G}$, is given by [26]

$$F[\tilde{\mathcal{G}}, \mathcal{G}] = \int d\psi \;\; \mathrm{Tr}\left[\mathcal{G}^{\dagger}(\psi)\tilde{\mathcal{G}}(\psi)\right]. \tag{2.32}$$

The noisy channel can be decomposed as follows: $\tilde{\mathcal{G}} = \Lambda \circ \mathcal{G}$ with Λ the error caused by implementing the gate $\mathcal{G}$.
The fidelity can be used as a metric to define the accuracy of the outcome of a quantum channel. A quantum measurement or positive operator-valued measure (POVM) E_m, is trace-preserving $\sum_m E_m = 1$ [11, 28]. An ideal POVM with eigenstates $|\psi_m\rangle$ is given by $E_m = |\psi_m\rangle\langle\psi_m|$. So in an ideal situation, the probability,

or measurement fidelity of state m can be calculated through the overlap of the measurement E_m and a pure state $|\psi\rangle\langle\psi|$ [7]:

$$P_m = \mathrm{Tr}\left[\,|\psi\rangle\langle\psi|\,E_m\right]. \tag{2.33}$$

Another way is to use the superoperator representation in equation 2.26, so a single qubit POVM [7] becomes

$$\langle\langle E_m| = \frac{1}{\sqrt{2}}\big(1, \mathrm{Tr}(E_m X), \mathrm{Tr}(E_m Y), \mathrm{Tr}(E_m Z)\big). \tag{2.34}$$

In experimental setups, measuring a single qubit in the $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$-basis results in a POVM consisting of a statistical mixture given by

$$E_0^e = (1-\epsilon_0)\,|0\rangle\langle 0| + \epsilon_1\,|1\rangle\langle 1| \tag{2.35}$$
$$E_1^e = \epsilon_0\,|0\rangle\langle 0| + (1-\epsilon_1)\,|1\rangle\langle 1|, \tag{2.36}$$

where ϵ_0 and ϵ_1 are the read-out errors of the single qubit eigenstates [7]. Subsequently, the measurement fidelity is given by

$$F_0 = Tr(E_0^e E_0) = Tr(E_0\,|0\rangle\langle 0|) = 1-\epsilon_0 \tag{2.37}$$
$$F_1 = Tr(E_1^e E_1) = Tr(E_1\,|1\rangle\langle 1|) = 1-\epsilon_1. \tag{2.38}$$

Note that the read-out errors describe the errors in the state projection. Hence, this error model is incomplete because the wrong mapping of the projection states is not taken into consideration. The readout apparatus can assign the wrong projection state. To read out a measurement, a device (readout apparatus) uses classical signals, such as currents, RF signals, etc., to map the projected states. Mapping the projection state to a wrong signal, or assigning the signal to a wrong binary value causes errors, identified as "mapping errors" [7]. The mapping errors are depicted with η where η_0 is the error to map $|0\rangle$ onto $|1\rangle$ and η_1 to map $|1\rangle$ to $|1\rangle$. Both errors, on the state projection and the mapping during the readout, can occur at the same time. Which results in a correct outcome. The measurement fidelity can be extended to

$$F_0 = (1-\epsilon_0)(1-\eta_0) + \epsilon_0\eta_1, \tag{2.39}$$
$$F_1 = (1-\epsilon_1)(1-\eta_1) + \epsilon_1\eta_0. \tag{2.40}$$

The gate sequences applied in the RB protocol in Chapter 5 are combinations of π- and $\pi/2$-pulse with rotations around the x- and y-axis

$$\mathcal{F} = \{I_x, I_y, X_\pi, X_{\pi/2}, Y_\pi, Y_{\pi/2}, -X_{\pi/2}, -Y_{\pi/2}\}, \tag{2.41}$$

where I_x and I_y are identity gates, performing a 2π-rotation about the x- or y-axis $I_x = X_{2\pi}$ and $I_y = Y_{2\pi}$. Using the superoperator formalism and the gate set, a complete basis of input states $\{\mathcal{F}_i|\rho_0^{ideal}\rangle\rangle\}$ and a complete measurement basis $\{\langle\langle E_0^{ideal}|\mathcal{F}_i\}$ can be set up. Therefore, we define a state and POVM as $\rho_0^{ideal} = |0\rangle\langle 0|$ and $E_0^{ideal} = |0\rangle\langle 0|$. Eventually, SPAM errors are defined as the errors making ρ_0^{ideal} and E_0^{ideal} imperfect:

$$\rho_0 = (1-\epsilon)\,|0\rangle\langle 0| + \epsilon\,|1\rangle\langle 1|\,, \tag{2.42}$$

$$E_0 = (1-\eta_0)\,|0\rangle\langle 0| + \eta_1\,|1\rangle\langle 1|\,, \tag{2.43}$$

where ϵ and $\eta_{0,1}$ represent the SPAM errors. The advantage of using an RB protocol is the ability to isolate SPAM errors and simultaneously characterize both SPAM errors and gate errors. The price paid is that only the average gate fidelity can be extracted by amplifying the gate errors in a depolarizing channel while leaving the SPAM errors constant.

Chapter 3

Theoretical approach of the Standard and Interleaved Randomized Benchmarking protocol

Chapter 2 covers the theoretical approach of the standard and interleaved randomized benchmarking protocol. First, we will dig deeper into twirling quantum channels and how we need them to extract the average gate fidelity of a quantum channel. The second part motivates the use of Clifford operators in the RB protocol and derives the decay parameters for zero- and first-order approximations of the error channel. At last, the mathematical insights shall be used to explain the SRB and IRB protocol.

3.1 The exact Haar Twirl

In quantum information theory, "twirling" refers to a technique used to analyze and simplify quantum channels. It could be anything from a simple quantum gate to a more complex operation like a noisy quantum channel [29, 30]. Twirling a quantum channel is essentially averaging it over certain operations to simplify its properties. The twirl performs an average over a set of operations that form a group, such as all possible unitary transformations or all possible permutations. This averaging process helps to understand the overall behavior of the channel and often simplifies calculations. Twirling is particularly useful in quantum information theory because it can help us identify symmetries and invariant properties of quantum channels. These symmetries can be exploited for various purposes, such as error correction, quantum state estimation, or characterizing the capacity of quantum communication channels. The exact Haar twirl is defined by

$$\begin{aligned}\Lambda^{ave} &= \int dU\ U^\dagger \circ \Lambda \circ U \\ &= \int dU\ U^\dagger \Lambda(U\left|\psi\right\rangle\left\langle\psi\right|U^\dagger)U \end{aligned} \tag{3.1}$$

where $\int dU\ = 1$, uniform distribution over unitaries, i.e., the Haar measure. The goal of twirling quantum channels in this thesis is to manipulate quantum operations so that a simple experiment can be used to extract this gate fidelity in a more efficient way than through QPT. By definition, the Haar measure (with respect to the unitary group) is a topological measure that defines a uniform distribution over Kraus operators [30].

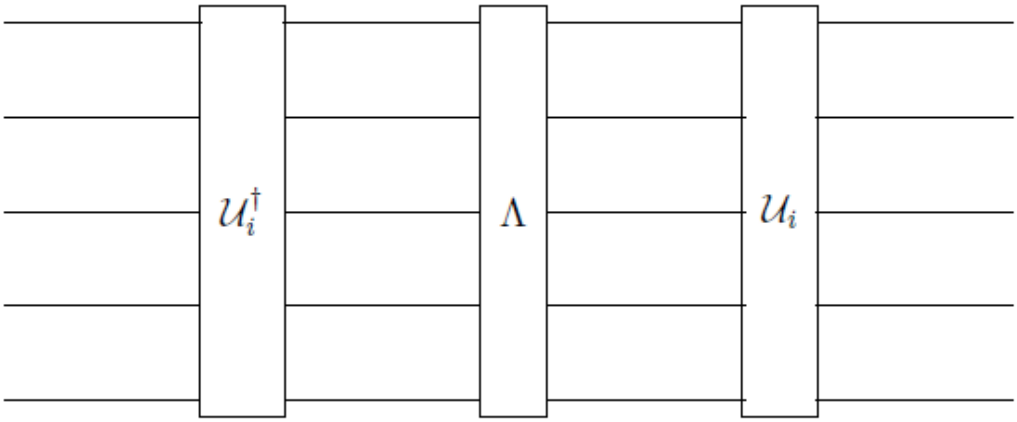


Figure 3.1: Discretized schematic representation of Twirling a quantum channel. The averaging can be written as $\overline{\Lambda}(\rho) = \sum_i pr(U_i)U_i^\dagger \circ \Lambda \circ U_i(\rho)$, where $pr(U_i)$ is the probability distribution over U.
Source: Magesan, E. et al. [29]

To see how the Twirl of a quantum channel is useful to extract the average gate fidelity, we follow the steps depicted in [30], and start from a quantum channel represented by the superoperator Λ_U and use Kraus representation to define this operator:

$$\mathcal{G} = \sum_i K_i \rho K_i^\dagger \tag{3.2}$$

Rewrite this by interleaving the identity operator twice using unitaries gives

$$\mathcal{G} = \sum_i (K_i U^\dagger) U \rho U^\dagger (U K_i^\dagger). \tag{3.3}$$

Next, we define a new operator

$$\Lambda = \sum_i (K_i U^\dagger) \otimes (U K_i^\dagger) = \sum_i K_i U^\dagger \otimes (K_i U^\dagger)^\dagger, \tag{3.4}$$

so that $\mathcal{G}(\rho) = \Lambda(U\rho U^\dagger)$, recall $\tilde{\mathcal{G}} = \Lambda \circ \mathcal{G}$ from the previous section. By defining this new operator Λ, the error that differentiates the operator (or map) $\mathcal{G}$ from the intended unitary operator U is encapsulated. The gate fidelity from equation 2.28 can now be rewritten as

$$F(U|\psi\rangle\langle\psi|U^\dagger, \mathcal{G}(|\psi\rangle\langle\psi|)) = F((U|\psi\rangle)(\langle\psi|U^\dagger), \Lambda((U|\psi\rangle)(\langle\psi|U^\dagger)), \tag{3.5}$$

the invariance of the Haar measure allows renaming the variable of integration from $U|\psi\rangle$ to $|\phi\rangle$, so that

$$F(U, \mathcal{G}) = F(I, \Lambda). \tag{3.6}$$

Where we showed that the fidelity of the of $\mathcal{G}$ and the ideal unitary operator U is equal to the fidelity of Λ, the superoperator that encapsulates the error, and the identity operator. Next, it's easy to verify that the fidelity of a superoperator Λ with the identity operator is the same as the fidelity of the exact Haar twirl of Λ with the identity:

$$\begin{aligned} F\Big(\int dU\ U^\dagger \circ \Lambda \circ U, I\Big) &= \int d\psi\ \operatorname{tr}\Big(\langle\psi|\Big(\int dU\ U^\dagger \Lambda(U|\psi\rangle\langle\psi|U^\dagger)U|\psi\rangle\Big)\Big) \\ &= \int dU \int d\psi\ \operatorname{tr}\big((\langle\psi|U^\dagger)\Lambda((U|\psi\rangle)(\langle\psi|U^\dagger))(U|\psi\rangle)\big) \\ &= \int d\phi\ \operatorname{tr}\big(\langle\phi|\Lambda(|\phi\rangle\langle\phi|)|\phi\rangle\big) \\ &= F(\Lambda, I), \end{aligned} \tag{3.7}$$

where the transformation is used again $d(U|\psi\rangle) = d(|\phi\rangle)$ according to the Haar measure. The average gate fidelity can be $\overline{F_g}$ can then be calculated by average the gate fidelity of $\mathcal{G}$ over the uniform unitary distribution:

$$\begin{aligned}
\overline{F}_g = \int dU\ F(\mathcal{G}, U) &= \int dU\ F(\Lambda, I) \\
&= \int dU\ F\Big(\int dU'\ U'^{\dagger} \circ \Lambda \circ U', I \Big) \\
&= \int dU\ F\Big((1-p)\rho + pI, I \Big) \\
&= \int dU\ \int d\psi\ \mathrm{tr}\left(\langle\psi| (1-p) |\psi\rangle \langle\psi| + p\frac{I}{d}) |\psi\rangle \right) \\
&= \int dU\ \left(1 - p + \frac{p}{d}\right) \\
&= 1 - \mathbf{p} + \frac{\mathbf{p}}{d}, \qquad (3.8)
\end{aligned}$$

where we used in the third line that the Twirl of a quantum channel induces a depolarizing channel [30], and $\mathbf{p} = \int dU\ p$ the average depolarizing parameter of the twirled channel. Remember, the depolarizing parameter indicates the probability that the quantum state is fully mixed, and with probability p - 1, the state is preserved. Hence, the depolarizing parameters indicate the infidelity of a quantum channel. When measuring a quantum channel, the fidelity is extracted. This means that the p-parameter of the fidelity decay is a measure of the fidelity of the channel. Therefore, we need to make the following substitution: $p_f = 1 - p$, and p_f is the depolarizing parameter indicating the probability of staying in the initial state (fidelity). The depolarizing channel can be rewritten as

$$\Lambda_{dep} = p_f \rho + (1 - p_f)\frac{I}{d}, \qquad (3.9)$$

and the average fidelity becomes

$$\overline{F}_g = p_f + \frac{1 - p_f}{d}. \qquad (3.10)$$

With the above derivations, we have shown that: **If a quantum channel behaves as a depolarizing channel, the average gate fidelity can be extracted.**

3.2 Unitary 2-designs

In quantum mechanics, the concept of a unitary t-design is a set of unitary operators with specific properties. When averaging (or twirling) over the unitary set, the twirl accurately approximates the average behavior of all unitary operators acting on a certain number of quantum bits or qubits [29, 31]. In simpler terms, a unitary t-design is a set of unitary operations that, when applied repeatedly to a quantum

state and averaged over, behaves similarly to the average behavior of all possible unitary operations on the same number of qubits. The twirl of a channel $\Lambda(\rho)$ over a discrete unitary 2-design ($U_i \in U$) can be written as

$$\begin{aligned}\Lambda(\rho) &= \sum_{i}^{|U|} p_i U_i^\dagger \circ \Lambda \circ U_i(\rho) \\ &= \sum_{i}^{|U|} p_i \Big(U_i^\dagger \Lambda(U_i \rho U_i^\dagger) \circ U_i \Big).\end{aligned} \tag{3.11}$$

Equation 3.8 stated that if a quantum channel behaves as a depolarizing channel, the average gate fidelity can be extracted. It seems that a twirl of $\Lambda(\rho)$ over the Haar measure is a depolarizing channel [26]. The proof of this statement is beyond the scope of the thesis.

3.3 The one qubit Clifford group $\mathcal{C}_1$

In general, the n-qubit Clifford group $\mathcal{C}_n$ is the normalizer of the Pauli group ($\mathcal{P}_n$) [26]:

$$\mathcal{C}_n = \{U \in U(2^n) | \sigma \in \pm P_n \Longrightarrow U\sigma U^\dagger \in \pm \mathcal{P}_n\}. \tag{3.12}$$

The above statement means that any one-qubit unitary operator with this property is an element of the Clifford group $\mathcal{C}$ up to a global phase $e^{i\phi}$. A normalizer maps elements of a certain group to it self under conjugation. The action of $U \in \mathcal{C}_1$ is completely determined by the images of X and Z, $UXU^\dagger$ and $UZU^\dagger$ must anti-commute. Let's start with X, the operator can go to any element of $\pm\mathcal{P}_1$, or all the six directions in three-dimensions. But Z can only go to $\pm\mathcal{P}_1 \setminus \{\pm UXU^\dagger\}$, or four directions in three-dimensions when one axis is already fixed. Hence $|\mathcal{C}_1| = 6 \times 4 = 24$. When starting in the ground or excited state of a qubit, Clifford rotations on the Bloch sphere always end up in an eigenstate of a Pauli operator. One-qubit gates of the Clifford group are basically combinations of π and $\pi/2$ pulses. In addition, the Hadamard (H) and phase gates H are generators of the Clifford group. This can easily be demonstrated: $HXH^\dagger = Z$, $HZH^\dagger = X$, and $SXS^\dagger = Y$, $SZS^\dagger = Z$. Hence, the H and S gates with their variants are also members of the Clifford group.

So, why Cliffords? First, the Gottesman – Knill theorem says that circuits of Clifford gates are easy to simulate classically [25]. What makes them convenient for data processing and analysis. Cliffords enables the scalability of RB experiments, where the inverse of the first m operators needs to be calculated efficiently. Second, the Clifford gates have the property that errors introduced during their application

do not propagate throughout the entire computation [30]. Errors tend to stay local in Clifford circuits. This property is fundamental to the efficiency of quantum error correction schemes based on the stabilizer formalism. In a Clifford circuit, all gates belong to the Clifford group, which consists of unitary operations that map Pauli operators to Pauli operators under conjugation. Since Pauli operators represent local errors (e.g., bit-flip, phase-flip, etc.) on individual qubits, Clifford gates transform these local errors into other local errors.

At last, the Clifford group is a unitary 2-design. The averaging or twirl over the Clifford group is discrete, and can be written as

$$\int dU\ U^{\dagger}\Lambda U \sim \frac{1}{K}\sum_{j} C_j^{\dagger}\Lambda C_j, \tag{3.13}$$

and $j = 1, ..., K$ where K is the number of gates in the Clifford group. Twirling a quantum operation over the Clifford group (or any unitary 2-design) produces a depolarizing channel:

$$\frac{1}{K}\sum_{j} C_j^{\dagger}\Lambda C_j \approx \Lambda(\rho) = p_f\rho + (1-p_f)\frac{I}{d}. \tag{3.14}$$

In the sections above we derived a formula for the average gate fidelity in function of the depolarizing parameter. The derivation was under assumption that the quantum channel was a depolarizing channel. **The twirl of a random generated Clifford circuit will always produce a depolarizing channel**. Which is the second important mathematical insight to understand the theoretical approach of the RB protocol.

3.4 The standard Randomized Benchmarking protocol

3.4.1 Expected fidelity decay for random Clifford sequences

The following section derives the fidelity decay for random generated Clifford circuits, and are based on the derivations of Easwar Magesan [32, 33, 34, 35]. Assume the superoperator $\Lambda_{i_j,j}$ the gate and time-dependent error for every Clifford gate, the error for each K_m sequence is given by

$$S_{i_m} = \bigcirc_{j=1}^{m+1}\left(\Lambda_{i_j,j}\circ C_{i_j}\right) \tag{3.15}$$

$$= \Lambda_{i_{m+1},m+1}\circ C_{i_{m+1}}\circ\Lambda_{i_m,m}\circ C_{i_m}\circ ...\circ\Lambda_{i_1,1}\circ C_{i_1} \tag{3.16}$$

Next, we define a new random superoperator

$$D_{i_j} = C_{i_{j+1}} \circ ... \circ C_{i_1}, \tag{3.17}$$

associated with a new random Clifford gate D. The new superoperator satisfies

$$C_{i_j} = D_{i_j} \circ D_{i_{j-1}}^{\dagger}, \tag{3.18}$$

since $C_{i_j} \circ C_{i_j}^{\dagger} = \mathbb{1}$. Inserting D in equation 3.15 gives

$$\begin{aligned} S_{i_m} &= \Lambda_{i_{m+1},m+1} \circ D_{i_{m+1}} \circ D_{i_m}^{\dagger} \circ \Lambda_{i_m,m} \circ D_{i_m} \circ D_{i_{m-1}}^{\dagger} \circ ... \circ D_{i_1}^{\dagger} \Lambda_{i_1,1} \circ D_{i_1} \\ &= \Lambda_{i_{m+1},m+1} \circ \bigcirc_{j=1}^{m} \Big(D_{i_j}^{\dagger} \Lambda_{i_j,j} \circ D_{i_j} \Big). \end{aligned} \tag{3.19}$$

Next, we introduce the average sequence operation

$$S_m = \int_{i_m,...,i_1} \frac{1}{K_m} S_{i_m} \approx \frac{1}{K_m} \sum_{i_m}^{K_m} S_{i_m},$$

where we use the discrete representation in the Clifford group. Starting in state $|\phi\rangle$, the fidelity of a single sequence is given by $\mathrm{Tr}[E_\psi S_{i_m}(\rho_\psi)]$, which equals the measurement fidelity from equation 2.33, where E_ψ is a POVM element. According to [32], a perturbative expansion of the error superoperator can be performed to isolate the mean value Λ from a small perturbative part

$$\Lambda_{i_j,j} = \Lambda + \delta\Lambda_{i_j,j}. \tag{3.20}$$

A zeroth-order approximation corresponds to $\Lambda_{i_j,j} = \Lambda$, where only gate and time-independent errors are considered. A first-order approximation yields one gate and time-dependent error in the m+1 sequence. When increasing the order to $k = m$+1, all the gate and time-dependent errors of the m+1 gates in the sequence are taken into account. The average sequence fidelity of the k-th order is given by

$$F_{seq}^{(k)}(m,\psi) = \mathrm{Tr}\left[\left(\sum_{k=0}^{m+1} S_{i_m} \right) (\rho_\psi) E_\psi \right]. \tag{3.21}$$

The zeroth-order approximation

Consider the zeroth-order approximation, where we assume gate and time-independent errors $\Lambda_{i_j,j} = \Lambda$. According to [33, 32, 35], the independent elements $(D_{i_j}^{\dagger} \circ \Lambda_{i_j,j} \circ D_{i_j})$ from equation 3.19 corresponds to the average channel superoperator, depicted in

equation 3.1. The averaged sequence in the zeroth-order approximation is therefore given by

$$S_m^{(0)} = \int_{i_m,\ldots,i_1} \frac{1}{K_m} \Lambda \circ \Lambda^{ave} \circ \ldots \circ \Lambda^{ave} \tag{3.22}$$

$$\approx \langle\langle E_\psi | \Lambda \circ (\Lambda^{ave})^{\circ m} | \rho_\psi \rangle\rangle, \tag{3.23}$$

where the superoperator Λ is absorbed into the measurement E_ψ [7]. Remember, that the average sequence operator results in the depolarizing channel operator $\lambda ave = \Lambda_{dep}$ for unitary 2-designs, such as the Clifford group. Eventually, the fidelity decay can be extracted by calculating the sequence fidelity of the average sequence in the zeroth-order approximation:

$$\begin{aligned} F_{seq}^{(0)}(m,\psi) &= \mathrm{Tr}\left[S_m^0(\rho_\psi) E_\rho\right] \\ &= \mathrm{Tr}\left[\Lambda \circ (\Lambda^{ave})^{\circ m}(\rho_\psi) E_\psi\right] \\ &= \mathrm{Tr}\left[\Lambda\Big(p\rho + (1-p)\frac{I}{d}\Big)^{\circ m} E_\psi\right] \\ &= \mathrm{Tr}\left[\Lambda\Big(p\rho + \frac{I}{d}\Big) p^m E_\psi\right] + \mathrm{Tr}\left[\Lambda\Big(\frac{I}{d} E_\psi\Big)\right] \\ &= A_0 p^m + B_0, \end{aligned} \tag{3.24}$$

where A_0 and B_0 absorb the SPAM errors. Note, that the depolarizing parameter extracts the fidelity and indicates the probability for the quantum channel to stay in the initial state. The substitution $p_f = 1$ - p is already considered in equation 3.24 with $p = p_f$.

Equation 3.24 shows the strength of RB protocols, the SPAM errors due to the imperfections of the initial state ρ_ψ and measurement E_ψ are encapsulated in the constants A_0 and B_0. In addition, the depolarizing parameter is isolated, not influenced by the SPAM errors. If there are no SPAM errors, the "visibility window" is given by $[B_0, A_0 + B_0] = [0.5, 1.0]$, graphically depicted in figure 3.2. Hence, the A_0 and B_0 parameters are an indication of the influence of SPAM errors or the quality of the experimental setup, which is one of the major advantages of RB compared with QPT.

Overall, the RB protocol can be summarized with three important mathematical insights:

1. If a quantum channel behaves as a depolarizing channel, the average gate fidelity can be extracted:

$$\overline{F}_g = p + \frac{1-p}{d}.$$

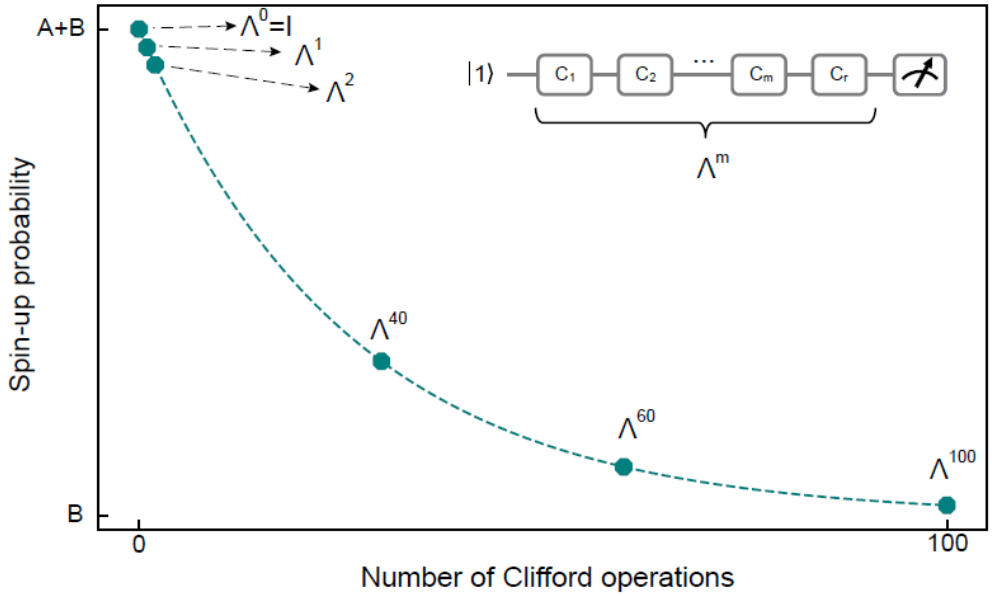


Figure 3.2: Zeroth-order approximation for the fidelity decay $F_{seq}^{(0)}(m,\psi) = A_0 p^m + B_0$. The fidelity of the initial state saturates to $B_0 = 0.5$ when the quantum state is a fully mixed state, due to the increasing number of depolarizing steps Λ^m . Source: Xue, X. (TUDelft) [7]

2. The twirled operation of a unitary 2-design, such as the Clifford group, is a depolarizing channel:

$$\frac{1}{K}\sum_j C_j^\dagger \Lambda C_j \approx \Lambda(\rho) = p\rho + (1-p)\frac{I}{d}.$$

3. Gate and time-independent errors lead to an exponential decay of the fidelity:

$$F_{seq}^{(0)}(m,\psi) = A_0 p^m + B_0 \tag{3.25}$$

where the depolarizing parameter p indicates the probability of staying in the initial state.

The first-order approximation

Let's add a gate and time-dependent error to the random generated sequence of m+1 Clifford gates. A gate-dependent error is fixed in the sequence on a certain gate. However, a time-dependent error propagates to different positions in the sequence. Hence, m+1 first-order perturbation terms are linked to the different positions of the

time-dependent error. We refer to *Easwar Magesan* [33] for the math, the first-order fidelity decay eventually becomes

$$F_{seq}^{(1)}(m,\psi) = A_1 p^m + B_1 + C_1(m-1)(q-p^2)p^{m-2}, \tag{3.26}$$

where A_1, B_1, and C_1 absorb the SPAM errors. The extra term q - p^2 can be seen as a measure of the gate dependence of the errors. Magesan stated that second-order approximations and higher can be neglected if the variation of the noise is smaller than $1/(m+1)$. In the next chapter, both zeroth and first-order approximation are compared to extract the average gate fidelity.

3.4.2 The standard Randomized Benchmarking protocol

The experimental implementation of an SRB protocol can be summarized as follows [15]:

1. For a set of sequence lengths $m \in \{m_1, m_2, ...m_M\}$

2. Generate n sequences (number of samples) of m Clifford gates and for each sequence $\mathcal{G} = \mathcal{G}_m ... \mathcal{G}_2 \mathcal{G}_1$:

   (a) Prepare the state $|\rho\rangle\rangle$ which is the pure state $|0...0\rangle$

   (b) Apply $\mathcal{G}$

   (c) Apply $\mathcal{G}_{inv} = \mathcal{G}^\dagger$

   (d) Measure $\langle\langle 0..0|\mathcal{G}_{inv}\mathcal{G}|\rho\rangle\rangle$ many times (k) to estimate the survival rate $F_{seq}^{(0,1)}(m,\psi)$. With k, the number of shots per sequence.

   The total random generated sequences N is therefore equal to $N = M \times n$

3. Average over all n samples for each length m to obtain $f(m)$

4. Fit the model $f(m) = Ap^m + B$ to extract $p = \frac{d.\overline{F}_g - 1}{d-1}$

The experimental procedure is schematically shown in figure 3.3. An RB experiment is nothing more than a Monte-Carlo simulation where a random sampling of Clifford sequences extracts the average gate fidelity.

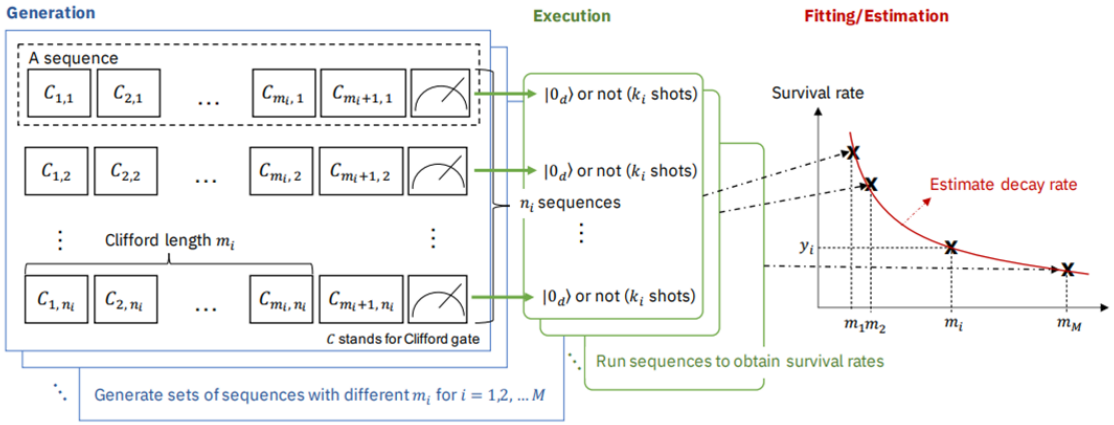


Figure 3.3: Experimental implementation of an RB protocol as mentioned in the text. Source: Itoko, T. et al. [15]

3.5 Interleaved Randomized Benchmarking

The last section covers the theoretical background of the interleaved RB protocol. The IRB is a useful tool to determine the estimated gate error of a certain Clifford operator [16, 17]. An IRB protocol is twofold: first, an SRB experiment is performed to extract the depolarizing parameter. Second, an IRB experiment is executed to extract the depolarizing parameter of the interleaved sequences with a certain Clifford gate, shown in figure 3.4. The ratio of both depolarizing parameters, gives us an estimation of the Clifford's gate error.

Assume $\mathcal{C}$ the gate we want to investigate (target gate). The first step is to interleave the target gate in the random generated sequences. The superoperator representing the interleaved sequence is given by

$$\mathcal{V}_{i_m} = \Lambda_{i_{m+1}} \circ \mathcal{C}_{i_{m+1}} \circ \Big(\bigcirc_{j=1}^{m} \left[\mathcal{C} \circ \Lambda_{\mathcal{C}} \circ \Lambda_{i_j} \circ \mathcal{C}_{i_j} \right] \Big), \tag{3.27}$$

where $\Lambda_{\mathcal{C}}$ is the superoperator encapsulating the error of the interleaved gate $\mathcal{C}$. Measuring each sequence $\mathrm{Tr}\left[E_\psi \mathcal{V}_{i_m}\right]$ gives the same fidelity decay from the previous section. Let $p_{\mathcal{C}}$ be the depolarizing parameter from the interleaved sequences. The estimated gate error of the target gate is therefore given as

$$r_{\mathcal{C}}^{est} = \frac{(d-1)(1-p_{\mathcal{C}}/p)}{d}, \tag{3.28}$$

which is equal to $r_{\mathcal{C}} = 1 - \overline{F}_{\mathcal{C}}$ and $\overline{F}_{\mathcal{C}}$ the average fidelity extracted through the interleaved sequences. According to [16], $r_{\mathcal{C}}^{est}$ must lie in the range $[r_{\mathcal{C}}^{est} - E, r_{\mathcal{C}}^{est} + E]$, and

$$E = min \begin{cases} \frac{(d-1)[|p-p_C/p|+(1-p)]}{d} \\ \frac{2(d^2-1)(1-p)}{pd^2} + \frac{4\sqrt{1-p}\sqrt{d^2-1}}{p}. \end{cases} \tag{3.29}$$

If Λ is depolarizing, $E = 0$, and above bounds are an overestimation of the gate error [16]. The simulations performed in the next chapter use the IRB protocol to estimate the gate error of a native gate in the quantum environment of the experiment.

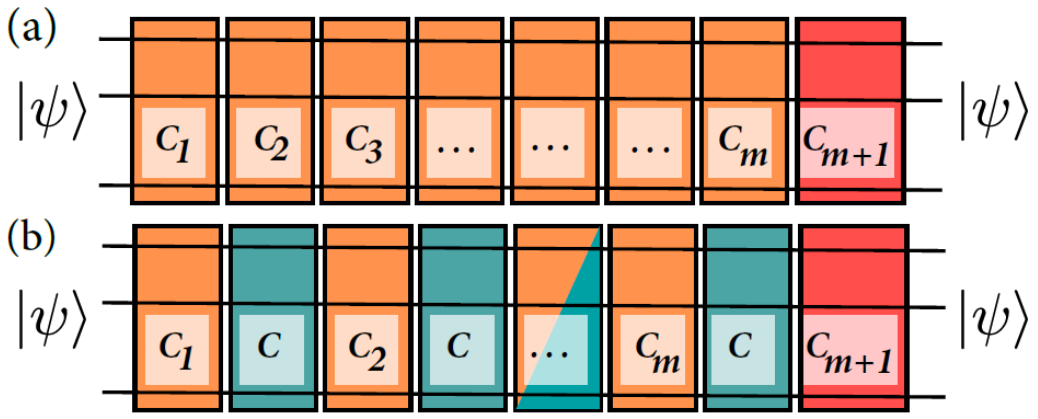


Figure 3.4: (a)-(b) Sequences for SRB and IRB protocols. (b) A target gate (green) is interleaved in random sampled Clifford gates (orange). The inverse gate (red) is attached at the end to return the initial state $|\psi\rangle$.
Source: Magesan et al. [16]

Chapter 4

Simulations of the Standard and Interleaved Randomized Benchmarking experiments

Chapter 4 covers the simulations of SRB and IRB protocols using the IBM quantum environment and Qiskit, an open-source Python package from IBM [18]. The first part of this chapter outlines the code we use to simulate the RB protocols. So, the results obtained can be scrutinized in the most transparent way. The simulations are executed exclusively in a local environment. The local environment is used to test and generate the RB protocol. Therefore, backends simulate a quantum environment on our personal computer. The IBM quantum environment also provides non-local backends to execute the code on real quantum computers at IBM's facilities, but we shall not use them due to the costs involved. The simulations aim to gain experience with RB protocols and Qiskit and see how a certain gate performs on a quantum computer.

4.1 Code outline

A python class *ONEQ_RBClass* executes the SRB and IRB experiments on a given backend and can be consulted in Appendix D.2. The following class parameters set up the experiment:

1. **circuit_lengths**:
   List the number of gates of each circuit m, e.g., $[m_1, m_2, m_3, m_4, m_5] = [5, 10, 50, 100, 250]$.

2. **sample_amount**:
   Number of sequences with a given length m that needs to be generated. Hence,

the total number of sequences N in an RB experiment is N = circuit_lengths $\times$ sample_amount = $M \times n$.

3. **seeds**:
   A list of length N, filled with integers, representing the generated sequences. Each seed value corresponds to a sequence of generated gates for a given random number generator, i.e., a seed. So, if you provide the same seed twice, you get the same sequence of quantum gates twice. In this way, each sequence is randomly composed and the Monte-Carlo simulation still holds so that each experiment can be repeated exactly. This gives a major advantage for researchers to repeat an RB experiment with different experimental parameters and conditions to see how the quantum computer behaves.

4. **initial_state**:
   To simulate different or mixed initial states of the quantum computer, the class parameter initial_state can be set as a Qiskit QuantumCircuit object to define the initial quantum state of each sequence. The quantum circuit depicting the initial state is added to the front of the randomly generated sequences.

5. **Backend**
   A backend is a code or server that represents the quantum environment of the quantum computer. In Qiskit, a backend can be used on local devices (personal computers) and IBM servers. The fake_provider package [**?**] contains backends (python files) to simulate the behavior of the real quantum computers. The Fake1Q is the default backend and mimics a one-qubit quantum computer. Hence, a small quantum computer can be simulated on a local device like our personal computer. A second option is to use the Aer package [36] to build a custom-made noise model. For example, it contains a depolarizing error, thermal relaxation error, etc. At last, the sequences can be executed on a real quantum computer by calling the IBM Runtime service [37]. The callable quantum computers depend on your current access plan at the IBM Quantum Platform. However, we are using the free access plan, and the RB experiment requires a lot of time due to the number of different circuits that need to be measured. Therefore, we shall not use the Runtime service, due to the limited access to the servers.

6. **interleaved**:
   Class parameter is a boolean. Set True to generate random sequences for an interleaved RB simulation.

7. **interleaved_gate**:
   Class parameter is a qiskit.circuit.library.standard_gates object that indicates the gate to be interleaved for IRB experiments.

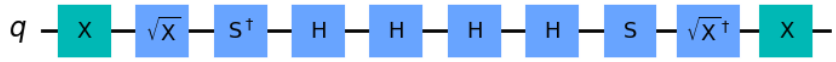
8. **state_fidelity**:
   Class parameter is a boolean. Set True to measure the process fidelity of the gate sequences instead of the shot measurement. Therefore, two density matrices are saved at the beginning and end of each sequence, following equations 2.23 and 2.27. Note that the cleanest way to describe a quantum state is through statevectors. However, we need noise models to simulate the quantum environment and the qubit's behavior, which leads to mixed states in the quantum channels/sequences. Hence, density matrices are used to calculate the process fidelity if asked.

The function *random_clifford_circuit* samples random Clifford gates and calculates the inverse circuit to attach at the end of the gate sequence. Then, Qiskit's transpiler, transpiles the generated circuit to a circuit that can be run on the given backend. The circuit is, therefore, decomposed into the backend's native gates. The transpiler is also an optimization, so the optimization level is set to zero to prevent the transpiler from returning one identity gate. Figure 4.1 shows the generated and transpiled quantum 5-gate circuit for the default Fake1Q backend.

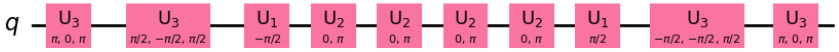
As shown in figure 4.1, a 5-gate circuit has actually ten gates. The reason for this is that Qiskit mirrors the random generated circuit. The first five gates are randomly chosen by a certain seed, the other five circuits mirror the first five. The length of each sequence doubles, so the outcome is always the initial state in a perfect simulation. In real RB experiments, just one $m+1$ gate is defined such that the circuit returns the initial quantum state. The set of gates to sample the random generated circuits consists of both Pauli's and generators of the Clifford group:

$$\mathcal{F} = \{X, Y, Z, H, S, S^\dagger, \sqrt{X}, \sqrt{X}^\dagger\}. \tag{4.1}$$

The matrix representation of the Clifford gates can be consulted in appendix A.1. An RB simulation can be set up by defining an *ONEQ_RBClass* Class object, with the corresponding class parameters to characterize the simulation and experimental environment. Eventually, the function *execute_randomized_benchmarking* samples and measures the sequences to gather the results. The measurement fidelity is the probability of measuring the initial state ($|0\rangle$-state by default) and is defined as the ratio of the number of counts the initial state returns over the number of shots.



(a) Random generated quantum circuit



(b) Transpiled quantum circuit

Figure 4.1: The Random quantum circuit is generated with lucky number seed = 42. The Fake1Q backend uses native gates: $U_1(\lambda)$=$U(0,0,\lambda)$, a single-qubit rotation about the Z axis. $U_2(\phi,\lambda) = U\left(\frac{\pi}{2},\phi,\lambda\right)$, a single-qubit rotation about the X and Z axis, and $U_3(\theta,\phi,\lambda) = U\left(\theta,\phi,\lambda\right)$, a generic single-qubit rotation gate with 3 Euler angles. For example, in this backend, the Hadamard gate (H) can be transpiled as H = $U_2(0,\pi)$. The matrix representation of these gates can be consulted in appendix A.3.

4.2 Simulations of standard RB experiments

4.2.1 The AER noise models

The first SRB simulation uses a noise model from the Aer package to build a custom noise model. The Aer custom noise backend transpiled the circuit in $U_{1,2,3}$ gates and preserves the X gate, depicted in figure 4.2.

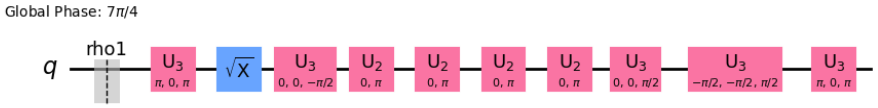


Figure 4.2: Transpiled circuit (seed = 42) with the Aer custom noise backend. Here, the X and even the $\sqrt{X}$ gate is a native gate and must not be transpiled. Similar to the Fake1Q backend, the Aer uses U_1, U_2, and U_3 gates. The "rho1" object at the beginning of the circuit is where the first density matrix is saved. The second density matrix is determined at the end of the circuit when the state fidelity is calculated.

To build up the noise model, the native gates U_2 and U_3 will be linked to certain errors. First, a depolarizing error with p = 0,01 is attached to the U_2 gate. So, with a one percent probability, the U_2 gate returns an identity gate (mixed state of gates). Second, a thermal relaxation error is linked to the U_3 gate. The thermal

relaxation error requires the T_1 time, T_2 time, and t gate time parameters, satisfying the condition $T_2 \leq 2T_1$. The noise model uses a $T1/T2 \sim 10^2$ ratio and the default gate time of 100. The code can be consulted in Appendix D.1.

The data acquired by running the experiment is given in appendix C.1, to illustrate how the returned data file could look like. The means of the samples associated with a certain circuit length are used to fit the zeroth and first-order approximation of the fidelity decay. Both the average fidelity and the average length of the transpiled circuits are determined within the samples to acquire those mean values. Remember the zeroth and first-order approximation is given by

$$F_{seq}^{(0)}(m) = A_0 p^m + B_0 \tag{4.2}$$
$$F_{seq}^{(1)}(m) = A_1 p^m + B_1 + C_1(m-1)(q-p^2)p^{m-2}. \tag{4.3}$$

To eventually determine the average gate fidelity $\overline{F}_g(\rho) = p + \frac{(1-p)}{d}$, propagation of uncertainty [38] is applied to calculate the accuracy of the average gate fidelity. The standard deviation is therefore given by

$$\begin{aligned} \sigma_F &= \left|\frac{d\overline{F}_g}{dp}\right|\sigma_p \\ &= \frac{\sigma_p}{2}, \end{aligned} \tag{4.4}$$

where $d = 2^n = 2$. Figure 4.6 shows the results and fits of the SRB simulation with the Aer custom noise model, fits are generated by the Scipy curve fitting tool. According to table 4.1, $p_0 < p_1$, the same holds for their accuracies, but the R^2-value of the first order approximation is lower, so might say also less reliable compared to the zeroth order approximation. Calculating the average gate fidelity for both approximations gives

$$\overline{F}_g^{(0)} = 0.99699 \pm 9.0e-5 \tag{4.5}$$
$$\overline{F}_g^{(1)} = 0.999982 \pm 1.5e-6 \tag{4.6}$$

Although both results look very similar, the fidelity obtained in the first-order approximation is an order of magnitude more accurate compared to the zeroth-order approximation. This would be an astonishing advancement of the device's accuracy in real experiments. The same can be done by obtaining the state fidelity, shown in figure 4.7. The results are gathered in table 4.2, where it is immediately noticeable that the first-order approximation gives a p-value equal to one. The state fidelity method is a more mathematical way of obtaining fidelity. Therefore, the outcome will be higher than the simulations where the fidelity is obtained by measuring the

quantum circuits. A $p = 1$ is only possible in pure mathematical simulations and is not physical, simulations where the circuits are measured will always result in $p < 1$ and $\overline{F}_g < 1$. Note, that the fidelity decay saturates after 600 native gates, where the final quantum state is a mixed state. The probability for measuring a 0 or 1 is equally distributed and therefore equals 0,5. The average gate fidelity eventually becomes

$$\overline{F}_g^{(0)} = 0.99695 \pm 9.0e - 5, \tag{4.7}$$

where the first-order approximation is left out, due to the fact this is a non-physical result.

Table 4.1: Zeroth and first-order fidelity decay approximation for measurement fidelity, for the Aer custom noise model.

Measurement fidelity			
order of approximation	p-value	p accuracy (std)	R^2-value
0th approx. (p_0)	0.99398	0.00017	0.999
1st approx. (p_1)	0.999964	0.000030	0.984

Table 4.2: Zeroth and first-order fidelity decay approximation for state fidelity, for the Aer custom noise model.

State fidelity			
order of approximation	p-value	p accuracy (std)	R^2-value
0th approx. (p_0)	0.99393	0.00018	0.999
1st approx. (p_1)	1.0	$\sim 10^{-11}$	0.983

The next question arises whether the std depends on the number of sequence lengths in the simulation. Let's run the same simulation with 13 sequence lengths ([5, 10, 25, 50, 75, 100, 150, 200, 250, 300, 350, 450, 600]) and seed = list(range(130)). The measurement fidelity returns p-values with higher accuracy, shown in table 4.3, although the R^2-value for the first-order fit is decreased. The noise model uses unitary matrices to calculate the final quantum state, where some errors are added with a certain probability. But, still, the required calculations are done with 2x2 matrices, in the one-qubit experiments. Our personal computers are really performant in computing with 2x2 matrices. The number of gates we could add in a sequence is almost endless, and so is the number of lengths. The point of these simulations is

to work with a fixed number of sequence lengths and seeds, so the simulations are comparable, and by tuning the noise parameters, we can see the simulated behavior of a quantum channel with a given error. In real devices, the thermal relaxation time (T_1) limits the number of gates that can be executed on the qubit. Where the accuracy can be increased by taking more intermediate lengths.

Table 4.3: Zeroth and first-order fidelity decay approximation for state fidelity, for the Aer custom noise model with 13 different sequence lenghts.

State fidelity			
order of approximation	p-value	p accuracy (std)	R^2-value
0th approx. (p_0)	0.99402	0.00013	0.999
1st approx. (p_1)	0.9999986	0.0000030	0.926

Noise model with p = 0,05 on SX and U1 gate and thermal relaxation error on U3

Another noise model where a 5% depolarizing error is attached to the SX and U1 gate. Along with the thermal relaxation error on the U3 gate from the previous simulation. The results for the zeroth order approximation are gathered in table 4.4. The first-order approximation is ignored due to the low R^2 value: $R^2 = 0{,}515$.

Table 4.4: Results for the zeroth fidelity decay approximation for measurement fidelity, for the Aer custom noise model with a 5% depolarizing probability on the X and U1 gate.

Measurement fidelity			
order of approximation	p-value	p accuracy (std)	R^2-value
0th approx. (p_0)	0.97754	0.00060	0.999

The average fidelity for the zeroth order approximation becomes

$$\overline{F}_g^{(0)} = 0.98877 \pm 3.0e - 4, \tag{4.8}$$

and so the average fidelity decreases with one order of magnitude (from 0,99 to 0,98) compared to the previous noise model. The zeroth order fidelity decay, depicted in figure 4.8, is clearly steeper compared to the first noise model in figure 4.6, which results in a lower p-value and hence, average fidelity.

FakeSherbrooke Backend

Let us investigate the average gate fidelity of a real quantum computer. The Sherbrooke, for example, is a real 127-qubit quantum computer, which can be used via the IBM quantum platform. The coupling map of the Sherbrooke is depicted in figure 4.4. The available quantum computers at the IBM quantum platform are all 127-qubit quantum computers. Therefore, we chose one of them to investigate on our personal computer via the FakeSherbrooke backend, a backend that mimics the behavior of the actual Sherbrooke quantum computer. The focus will be on the performance of the first qubit since the SRB is a one-qubit protocol. The other 126 qubits are kept idle and are so-called ancilla qubits (or auxiliary qubits), which do not depend on the input state.

The fakeSherbrooke backend transpiles the seed = 42 circuit in native gates: X, SX and Rz (shown in figure 4.3). When performing an extensive RB experiment with parameters: lengths = [5, 10, 50, 100, 150, 200, 250, 300, 350, 450, 600]; samples = 20 and seeds = list(range(220)). The lowest measured fidelity is in the range of 0,86 - 0,88 as seen in figure 4.9, and the expected saturation point at 0,5 can not be reached. Because the backend represents a real quantum computer, the fidelity can only be extracted through measuring the circuit.

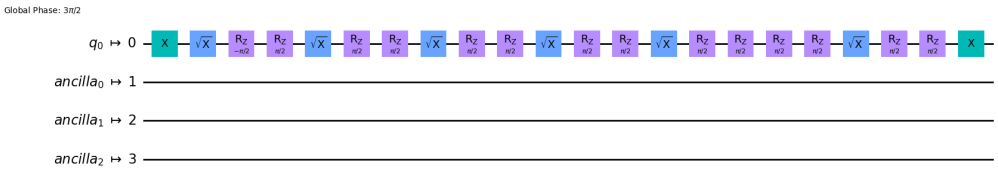


Figure 4.3: Transpiled circuit for seed = 42, with the FakeSherbrooke backend. Only the first four qubits are shown, other ancilla qubits are ignored.

Table 4.5: Zeroth and first order approximation for the FakeSherbrooke backend.

Measurement fidelity			
order of approximation	p-value	p accuracy (std)	R^2-value
0th approx. (p_0)	0.9999684	0.000028	0.999
1st approx. (p_1)	0.9999	0.0037	0.999

The p-values are gathered in table 4.5. Both approximations gives a p-value in the order of 0,9999 which is much higher compared to the noise models, what we expected from the fidelity decay in figure 4.9. Along with the high p-values,

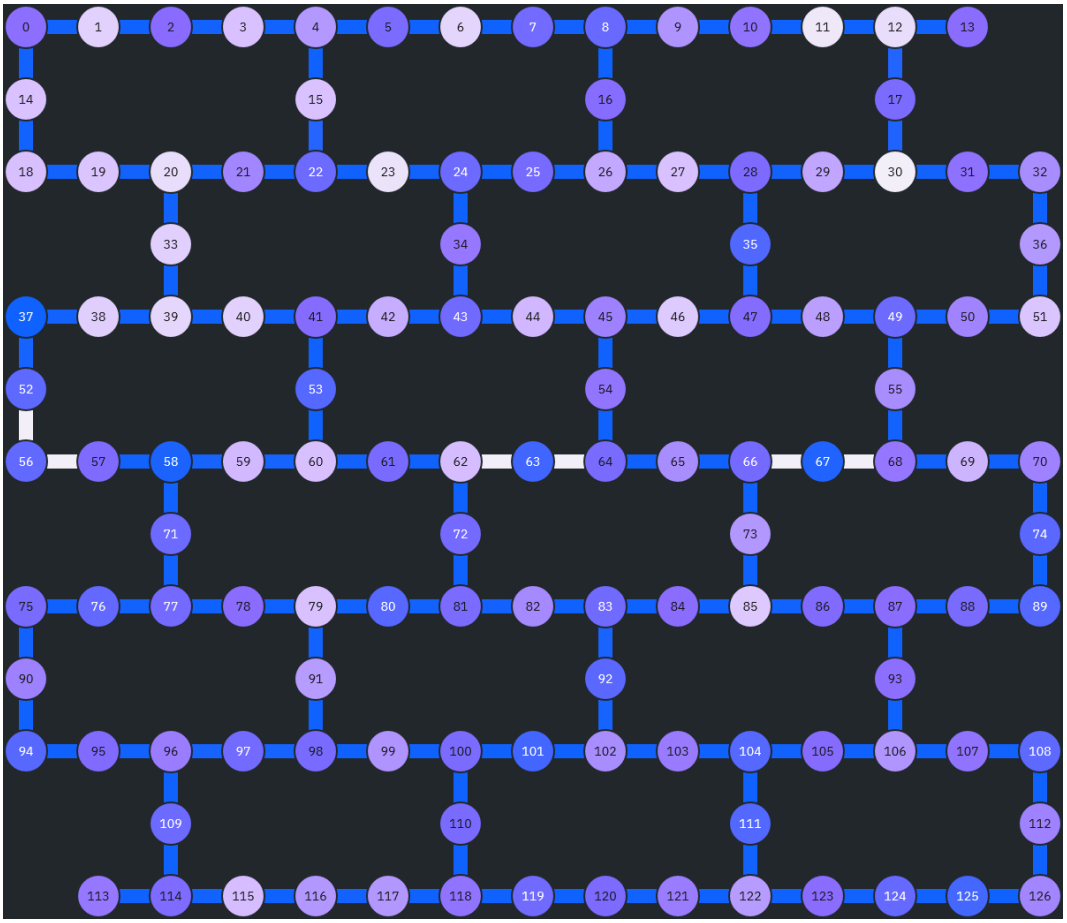


Figure 4.4: Coupling map of IBM's Sherbrooke 127-qubit quantum computer. The color of the qubits represents the T_1 time. For the first qubit q(0): $T_1 = 482{,}76\ \mu s$. The connections between the qubit give an indication of the ECR error. The ECR gate is an echoed RZX($\pi/2$) gate. This is equivalent to a CNOT up to single-qubit pre-rotations, where the echoing procedure mitigates some unwanted terms in an experiment. Source: IBM quantum [18]

the R^2-value for both approximation equals 0,999 which results in a high accuracy, especially for the zeroth order approximation. The average gate fidelity becomes

$$\overline{F}_g^{(0)} = 0.9999842 \pm 0.000014 \tag{4.9}$$

$$\overline{F}_g^{(1)} = 0.9999 \pm 0.0019, \tag{4.10}$$

where the accuracy for the zeroth order approximation is lower than the returned digits for the p-value in the Scipy package. It seems that the Sherbrooke is an extremely performant quantum computer. The next section will again rely on noise

models to determine the gate error of a specific gate.

4.3 Simulations of interleaved RB experiments

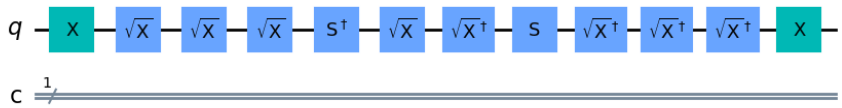
An Interleaved RB experiments aims to extract the estimated gate error $r_{\mathcal{C}}^{est}$ of a target gate $\mathcal{C}$. The sequences of quantum gates are interleaved with the target gate, to perform an RB experiment on the interleaved sequences. The ratio of the p-values from the SRB and IRB simulations can be exploited to estimate the gate error of the target gate given by

$$r_{\mathcal{C}}^{est} = \frac{(d-1)(1-p_{\mathcal{C}}/p)}{d}, \tag{4.11}$$

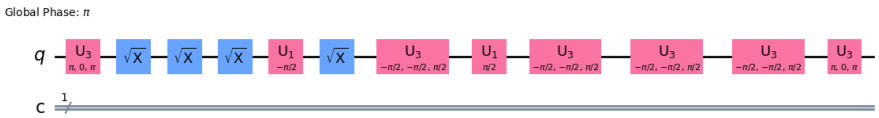
within the range [$r_{\mathcal{C}}^{est}$ - E, $r_{\mathcal{C}}^{est}$ + E], as seen in chapter 3. The noise model with the 5% depolarizing error on the SX and U1 gate, and thermal relaxation error on the u3 gate is used to simulate the effectiveness of IRB experiments.

Interleaved gate: SX

First, The SX gate is interleaved in the same sequences from the standard RB experiments. An example of an interleaved sequence and transpiled sequence is given in figure 4.5.



(a) Interleaved circuit with target gate: SX



(b) Transpiled quantum circuit

Figure 4.5: Interleaved quantum circuit and transpiled circuit with seed = 42. The SX gate is a native gate in the Aer package.

Table 4.6: Zeroth approximation for the IRB simulation with target gate: SX.

Measurement fidelity			
order of approximation	p-value	p accuracy (std)	R^2-value
0th approx. (p_0)	0.98715	0.00051	0.999

The results are gathered in 4.6 and depicted in figure 4.10. We only consider the zeroth order approximation to be comparable with the SRB experiment with the same noise model. The decay parameter is higher compared with the standard RB simulation, which indicates that the thermal relaxation error is the most dominant error in the circuits. The estimated error rate is $r_{\mathcal{C}}^{est}$ = -0,0049 with bounds [-0,021; 0,011]. A negative error rate is, of course, not a physical result. The depolarizing parameters in real IRB experiments, seen in the literature, are always lower compared to the SRB experiments. Although a specific gate can be really performant compared to the other gates in the experimental setup. I assume that the randomization (and the Monte-Carlo sampling) of the sequences, in the averaging (or twirl), results in a proper depolarizing channel, where the depolarizing parameter is higher compared with interleaved sequences. However, this is a topic that needs more scrutiny in further research. Here, in a custom noise model, the specific gate errors can differ from each other in a way the error rate for a low-error gate becomes negative.

Interleaved gate: S

The S gate is not a native gate in the backend, the gate is transpiled in a U3(θ, φ, λ) = U3($-\pi/2, -\pi/2, \pi/2$). The fidelity decay is shown in figure 4.11. The decay parameter in the zeroth order approximation is shown in table 4.7, where the p-value is lower compared to the SRB simulation. The fidelity decay is depicted in figure 4.11. Executing the IRB simulation results in an estimated error rate of = 0,0066 with bounds [0; 0,024]. To conclude, the U3 has a maximum estimated error rate of 2,4 %.

Table 4.7: Zeroth approximation for the IRB simulation with target gate: S.

Measurement fidelity			
order of approximation	p-value	p accuracy (std)	R^2-value
0th approx. (p_0)	0.9646	0.0015	0.999

4.4 Simulations: Fidelity decays

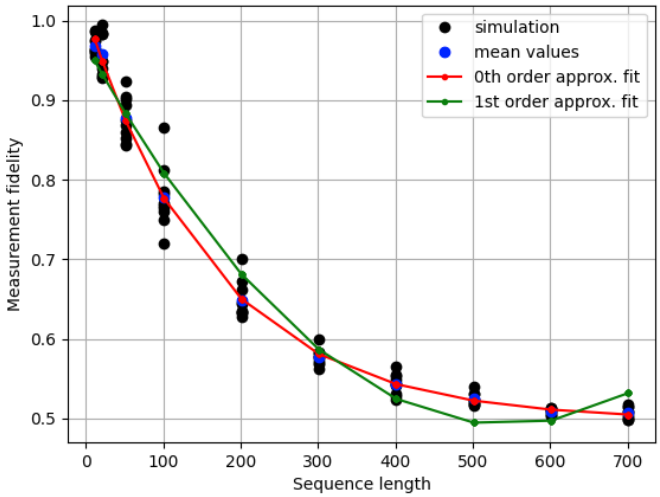


Figure 4.6: SRB simulation. Noise model with $p = 0{,}01$ on U2 and thermal relaxation error on U3. Measurement fidelity. Simulation parameters: lengths = [5, 10, 25, 50, 100, 150, 200, 250, 300, 350]; samples = 10; seeds = list(range(100))

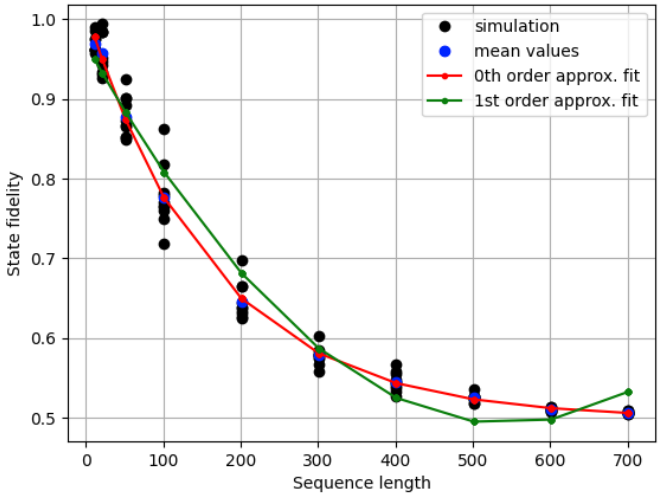


Figure 4.7: SRB simulation. Noise model with $p = 0{,}01$ on U2 and thermal relaxation error on U3. State fidelity. Simulation parameters: lengths = [5, 10, 25, 50, 100, 150, 200, 250, 300, 350]; samples = 10; seeds = list(range(100))

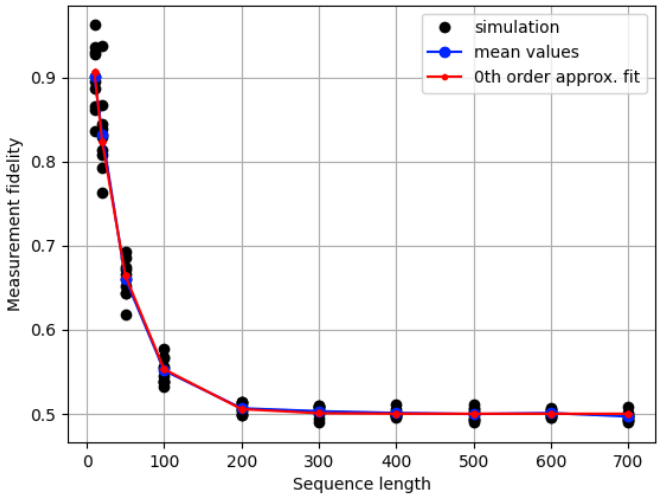


Figure 4.8: SRB simulation. Noise model with $p = 0{,}05$ on U1 and SX, thermal relaxation error on U3. State fidelity. Simulation parameters: lengths = [5, 10, 25, 50, 100, 150, 200, 250, 300, 350]; samples = 10; seeds = list(range(100))

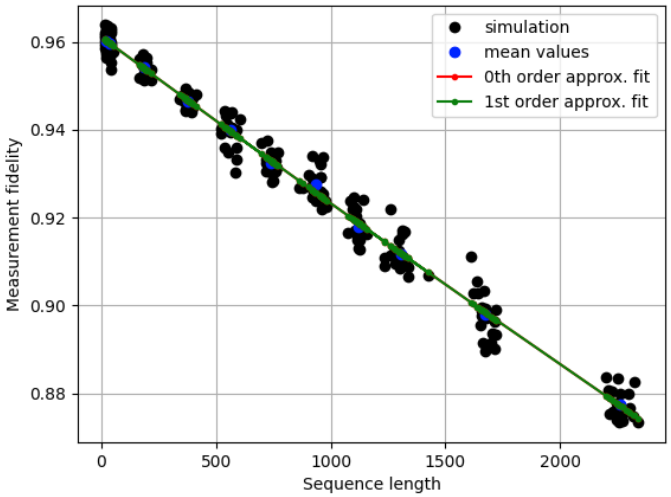


Figure 4.9: SRB simulation. FakeSherbrooke backend with simulation parameters: lengths = [5, 10, 50, 100, 150, 200, 250, 300, 350, 450, 600]; samples = 20; seeds = list(range(220))

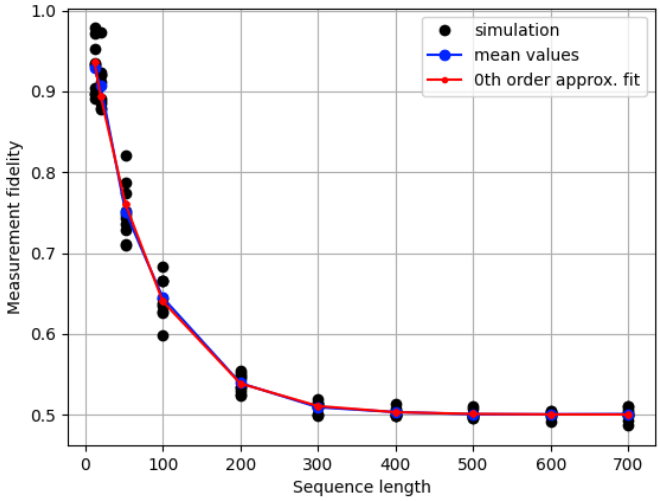


Figure 4.10: IRB simulation on SX gate. Noise model with $p = 0{,}05$ on U1 and SX, thermal relaxation error on U3. State fidelity. Simulation parameters: lengths = [5, 10, 25, 50, 100, 150, 200, 250, 300, 350]; samples = 10; seeds = list(range(100))

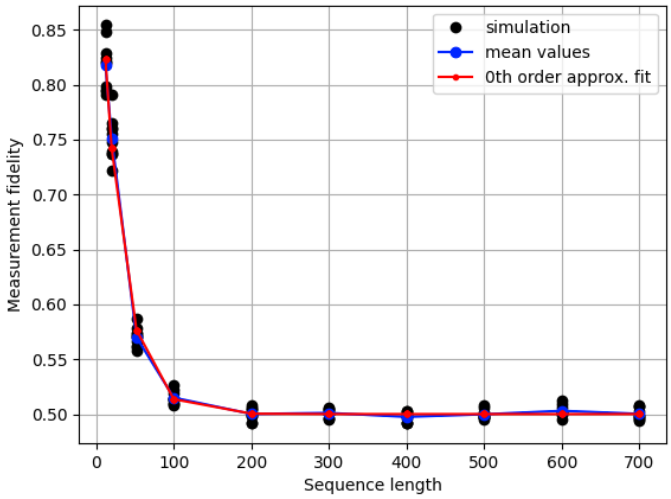


Figure 4.11: IRB simulation on S gate. Noise model with $p = 0{,}05$ on U1 and SX, thermal relaxation error on U3. State fidelity. Simulation parameters: lengths = [5, 10, 25, 50, 100, 150, 200, 250, 300, 350]; samples = 10; seeds = list(range(100))

Chapter 5

Practical implementation of a Randomized Benchmarking protocol

The last chapter covers the practical implementation of an RB protocol. From a technical overview of the gate-controlled silicon quantum dots to the quantum implementation of electron spin resonance, to link the hardware to the gate set. Eventually, IQ-mixing provides the last step to implement a gate sequence on a real quantum device. The ultimate goal is to have an RB protocol generating a random sequence of gates with given lengths, where the last gate puts the quantum state back in the initial state. That can be executed on the quantum hardware at Imec.

5.1 The quantum dot

A basic understanding of gate-controlled silicon quantum dots (SiQDs) is necessary to understand the steps to go from SiQDs to full working qubits. The following section briefly overviews the physics behind a gate-controlled quantum dot and how it's used to extract the spin-up fraction of an isolated electron. A Schematic circuit of a single quantum dot is shown in Figure 5.1. A gate-controlled quantum dot consists of a dot in the middle on top of the gate junction. The Fermi energy, i.e., the chemical potential, can be tuned through the (plunger) gate [19]. The source/drain junctions are located on opposite sites of the dot. When the Fermi energy matches the quantum dot's energy levels, a current flows through the dot. Hence, the current can be changed in the dot by tuning the gate voltage.

The induced charge in the dot is given by the product of the dot's capacitance and potential difference. A capacitance matrix C_{ij} relates the induced charge on each of the electrodes to the potential difference V_j:

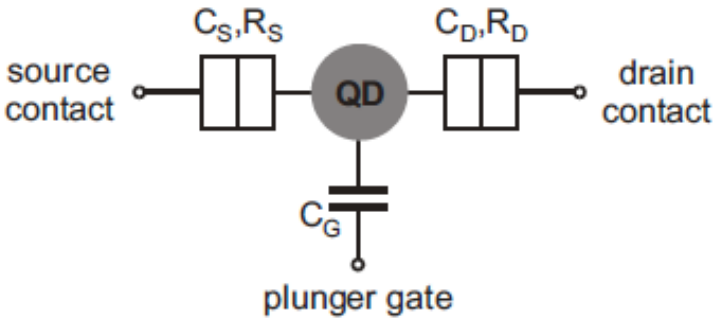


Figure 5.1: Schematic circuit of a single quantum dot, with the capacitance $C_{S,D}$ and resistance $R_{S,D}$ of the source and drain. The plunger gate, with capacitance C_G tunes the Fermi level, i.e., the chemical potential in the dot.
Source: Fuhner, C. (University of Hannover) [39]

$$Q_i = \sum_{j=0}^{n} C_{ij} V_j. \tag{5.1}$$

The j indices relate to the electrodes. The quantum dot, source, drain, and gate are denoted as $j = 0$, 1, 2, and 3, respectively. According to [40, 41] the electrostatic potential is given by

$$V_0(Q_0) = \frac{1}{C_\Sigma}\left(Q_0 - \sum_{j=1}^{n} C_{0j} V_j\right). \tag{5.2}$$

The total capacitance is defined as $C_\Sigma = \sum_{j=1}^{n} C_{0j}$. The induced charge $Q = -eN$, with N the number of electrons in the dot. The electrostatic energy of an isolated dot $U(N)$ can be found by taking the integral of the electrostatic potential:

$$U(N) = \int_0^{-eN} V_0(Q_0)\, dQ_0 = \frac{(eN)^2}{2C_\Sigma} + eN\left(\sum_{j=1}^{n} \frac{C_{0j}}{C_\Sigma} V_j\right), \tag{5.3}$$

where the first term is due to Coulomb repulsion, the second is the electrostatic energy of charges in the potential. The total energy $E(N)$ is the sum of the single-particle energies ε_i and the electrostatic energy and is given by

$$E(N) = \sum_{i=1}^{N} \varepsilon_i + \frac{(eN)^2}{2C_\Sigma} + eN\left(\sum_{j=1}^{n} \frac{C_{0j}}{C_\Sigma}\right), \tag{5.4}$$

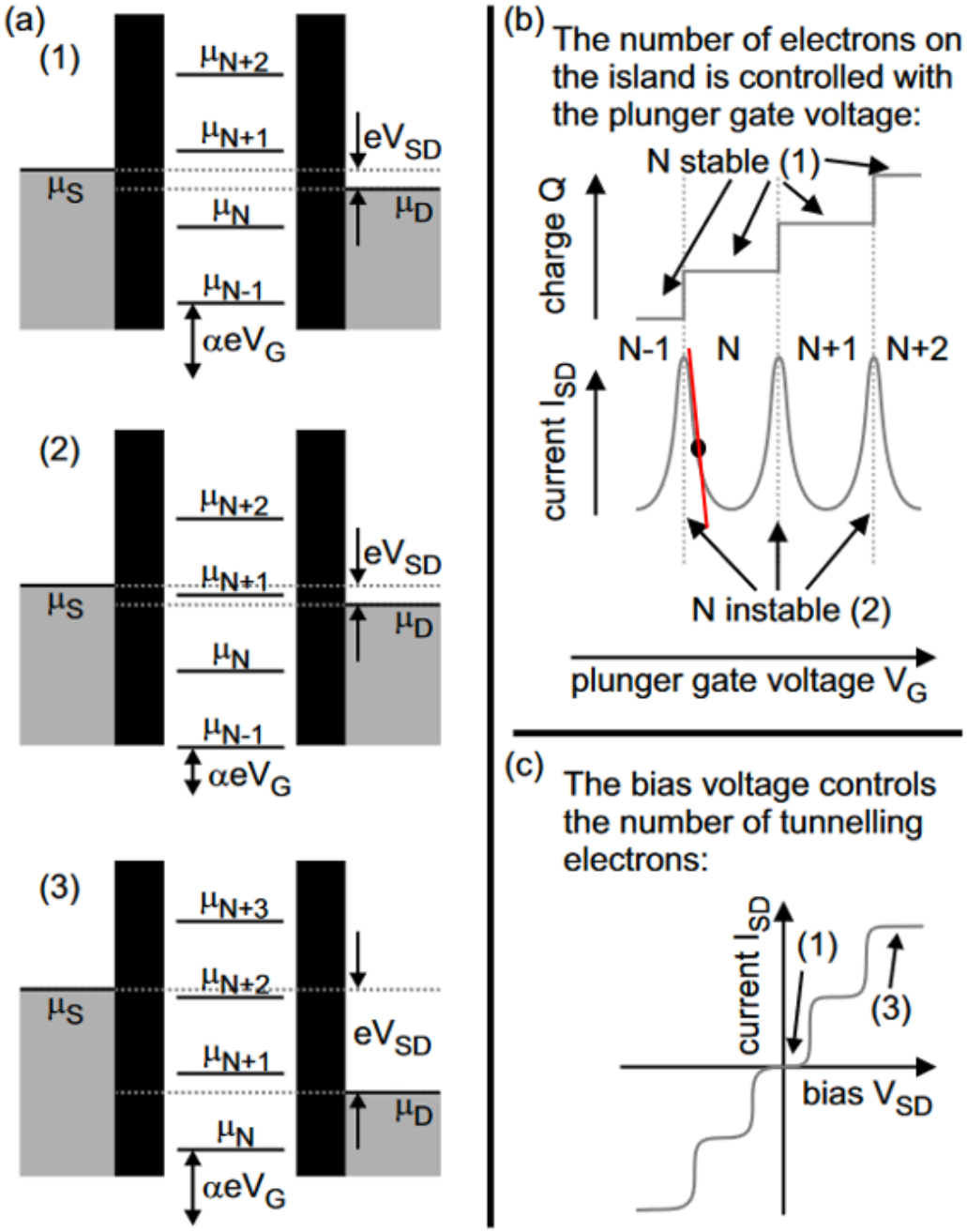


Figure 5.2: Schematic overview of the coulomb blockade. a) The chemical potential can be shifted with respect to the source/drain potential by alternating the gate voltage V_G. b) Coulomb resonances and blockades in function of the gate voltage. The plunger voltage, resulting in current I_{SD} at the point with the red slope, depicts the potential state in the SET. c) A Coulomb staircase is the result of the variation of the bias voltage V_{SD}.
Source: Fuhner, C. (University of Hannover) [39]

according to the constant interaction model [40]. Next, the chemical potential is defined as the energy needed to add the Nth particle to the dot

$$\mu_N = E(N) - E(N-1) = \epsilon_N + \frac{e^2}{C_\Sigma}(n - 1/2) + e\left(\sum_{j=1}^{n} \frac{C_{0j}}{C_\Sigma} V_j\right), \tag{5.5}$$

where we assume the difference between single-particle energies ε_{N+1} - ε_N can be ignored. At last, the energy to add one electron to the dot is given by the difference of μ_{N+1} and μ_N:

$$\mu_{N+1} - \mu_N = \frac{e^2}{C_\Sigma}. \tag{5.6}$$

Due to the coulomb blockade, a quantum dot can only contain discrete levels (i), containing $N + i$ electrons. A schematic representation is depicted in Figure 5.2. By variating the gate voltage, the number of electrons can be tuned to a stable situation where there is no current and N electrons are captured in the dot.

5.2 A physical qubit

SiQDs provide a platform for isolating and manipulating electrons and are the fundamental devices for creating physical qubits or spin qubit devices. The qubit $|0\rangle$- and $|1\rangle$-states are obtained by applying an external magnetic field $\mathbf{B} = B\hat{e}_z$. The Hamltonian of an electron due to the electron spin in an external magnetic field is given by $\hat{H}$ = -$\hat{\mu}$.$\mathbf{B}$, and $\hat{\mu}$ = -$g_e\mu_B\mathbf{S}$ the magnetic (dipole) moment, where g_e is the electron g-factor and $\mu_B = \frac{e\hbar}{2m_e}$ the Bohr magneton. The direction of the electron spin is opposite to the direction of the magnetic dipole moment. The single electron spin Hamiltonian can be rewritten as $\hat{H} = \left(\frac{g_e e\hbar}{2m_e}\right)\mathbf{B}.\mathbf{S} = \hat{\omega}.\hat{S}_z$, where $\hat{\omega} = \frac{g_e eB}{2m_e}\hat{e}_z$ and $\hat{S}_z = \frac{1}{2}\hbar\hat{\sigma}_z$. A general expression for the energy of an electron in an external magnetic field (also known as Zeeman splitting) can be written as

$$E = g_e\mu_B m_s B = \pm\frac{1}{2}\hbar\omega, \tag{5.7}$$

where $m_s = \pm 1/2$ is the spin magnetic quantum number, according to [1, -1] the eigenvalues of the $\hat{\sigma}_z$-operator. The lowest energy level, m_s = -1/2, is obtained when the magnetic field is aligned with $\hat{\mu}$ and counter-aligned with $\mathbf{S}$. The lowest energy state is therefore referred to as the spin-down or $|0\rangle$-state (the ground state). When $m_s = 1/2$, the electron spin is aligned with the magnetic field and counter-aligned with $\hat{\mu}$, which results in the highest energy state. The $m_s = 1/2$ energy state is sometimes referred to as the spin-up or $|1\rangle$-state. A schematic representation of a

physical qubit is depicted in figure 5.3.

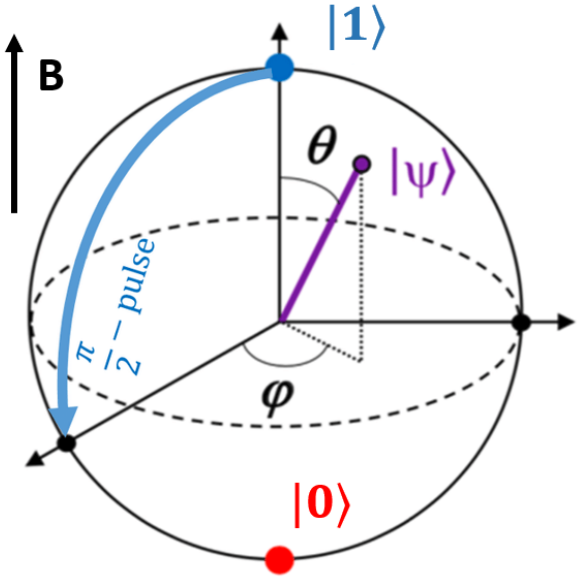


Figure 5.3: Schematic representation of a physical qubit in SiQDs. The $|1\rangle$-state or spin-up state is aligned with the external magnetic field. The $|0\rangle$-state is counter-aligned and therefore the ground state. A $\pi/2$-pulse brings the qubit in superposition where $|\Psi\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. Adapted from: Beckers, A. (ResearchGate) [42].

As discussed in the previous section, changing the gate voltage shifts the available energy levels. When a block curve with alternating a positive and negative period is applied to the gate. The positive voltage period induces the migration of electrons from the reservoir to the dot, i.e., loading the dot. When the negative voltage is applied, only spin-up particles in the $|1\rangle$-state can migrate back to the reservoir, as shown in Figure 5.4. This results in a displacement of charge which induces a change of current that can be measured through a Single Electron Transistor (SET).

A SET, as shown in figure 5.5, serves as a charge sensor. A distinction is made between the plunger voltage (V_p) and the top gate voltage (V_g). The plunger voltage regulates the chemical potential in the quantum dot and, hence, captures an isolated electron in the spin-up or spin-down state. The top gate controls the chemical potential of the reservoir and thereby determines the number of electrons that can accumulate on the SET. The gate voltage is chosen to induce a current in the SET with a steep slope, depicted in figure 5.2 b). Therefore, the sensitivity of the current through the SET for charge displacement is set maximally. The SET becomes a highly performant charge sensor, enabling the detection of individual electrons.

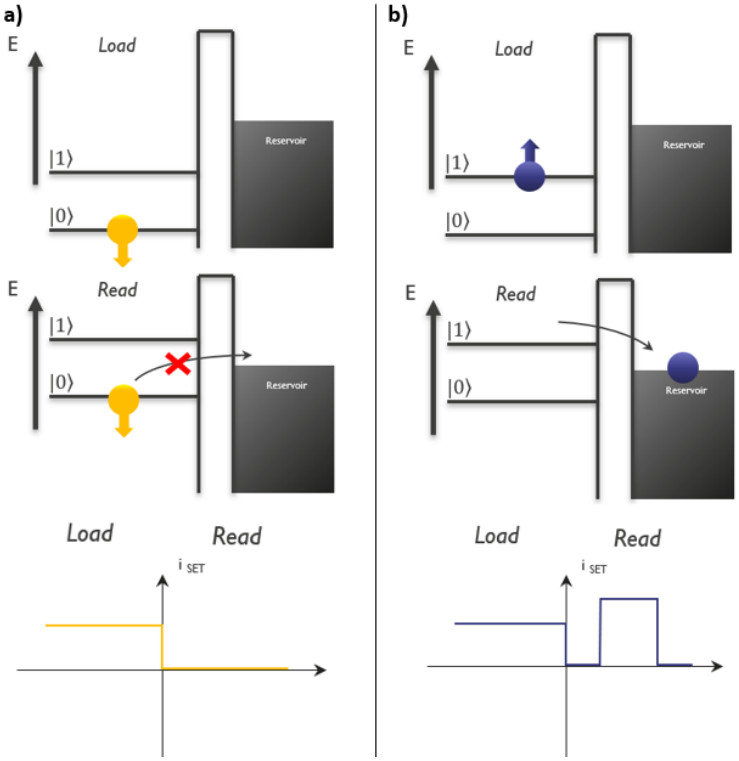


Figure 5.4: Graphical representation of the load and read period of the quantum dot. a) The energy level of spin-down ($|0\rangle$) particles is too low to migrate back to the reservoir. No current is measured. b) The migration of spin-up ($|1\rangle$) particles to the reservoir in the read period results in a current, measured in a SET.

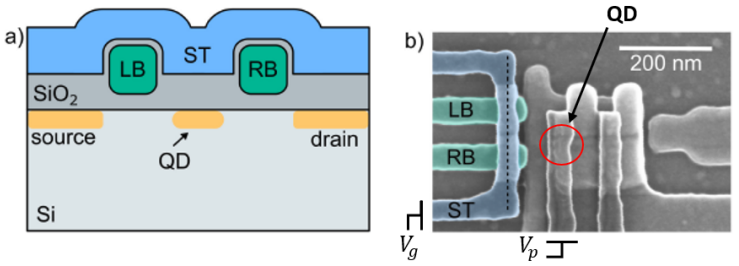


Figure 5.5: The single electron transistor (SET). a) Schematic representation of a SET with the quantum dot (QD) in the middle and left and right barrier gates on both sides (LB and RB). On to the top gate (ST). b) Top-down false-color SEM image of the charge sensor. The top gate accumulates the charge (V_g), and (V_p) regulates the energy levels in the quantum dot. The location of the quantum dot is indicated with the red circle. Adapted from: Dumoulin Stuyck, N. (KUL) [41]

5.3 Electron spin resonance: Applying quantum gates

So far, an electron with a specific quantum state is captured in a quantum dot, of which the spin-up fraction can be measured. The next step is to manipulate the isolated electron, in a way quantum gates can be performed on our qubit. Coherent driving rotates the qubit around the (x,y)-plane (the polar angle), so the qubit goes from the spin-up to the spin-down state and back. In this way the qubit can be brought into superposition, or a Z-gate (spin-flip) can be applied. The electron spin resonance (ESR) technique is a useful tool to perform coherent driving on an isolated electron and also provides the possibility to perform phase shifts in the (x,y)-plane, such that the axis of rotation can be varied to perform other gates such as X- and Y-gates. The derivations below show mathematically the necessary steps to perform coherent driving on a qubit. The previous section derived the energy and Hamiltonian for an isolated electron, where the Hamiltonian

$$\hat{H} = -\frac{\hbar\omega\hat{\sigma_z}}{2} = -\frac{\hbar\omega}{2}(|0\rangle\langle 0| - |1\rangle\langle 1|), \tag{5.8}$$

depicted the energy for an electron in an applied magnetic field. According to the Schrödinger equation from 2.3 and the unitary propagator in equation 2.5, the time evolution of a state $|\psi(t)\rangle$ of a closed quantum system is described by

$$|\psi(t)\rangle = e^{-iHt/\hbar}\,|\psi(0)\rangle\,, \tag{5.9}$$

so the initial state of the qubit is given by $|\psi(0)\rangle = e^{\frac{i\omega t}{2}}\,|0\rangle$, and can be written in the Bloch sphere representation from equation 2.2, with $\theta = \pi/2$ and $\varphi = -\omega t$. The Hamiltonian represents a rotation around the $\hat{e}_z$-axis with Lamor frequency ω [43, 44, 5]. The energy difference between the two states is given by

$$\Delta E = \hbar\omega = g_e\mu_B B_0, \tag{5.10}$$

where $\omega = \frac{g_z\mu_B B_0}{\hbar}$. Rotations between the $|0\rangle$- and $|1\rangle$-states are performed by applying ΔE, the resonance energy of the spin qubit. The ESR technique uses an oscillating magnetic field to generate radiofrequency waves with energy equal to the resonance energy ΔE, so the qubit starts rotating. The oscillating magnetic field is given by

$$\mathbf{B_1} = B_1\cos(\omega_p t + \phi_p)\hat{e}_x \tag{5.11}$$

where ω_p is the magnetic field frequency and ϕ_p a phase. Total Hamiltonian from the previous section can be written as:

$$\begin{aligned}\hat{H} = -\mu.\mathbf{B} = |\gamma|\mathbf{S}.\mathbf{B} \\ &= |\gamma_0|\hat{S}_x B_z + |\gamma_1|\hat{S}_x B_x \\ &= \frac{1}{2}|\gamma_0|B_z\hat{\sigma}_z + \frac{1}{2}|\gamma_1|B_1\cos(\omega_p t + \phi_p)\hat{\sigma}_x \\ &= \frac{\hbar\omega}{2}\hat{\sigma}_z + \frac{\hbar\omega_1}{2}\cos(\omega_p t + \phi_p)\hat{\sigma}_x \end{aligned} \tag{5.12}$$

where $\omega_1 = \frac{g\mu_1 B_1}{\hbar}$. The oscillating magnetic field can be generated through a radio wave antenna next to the SET. The Hamiltonian is twofold: the first part of the Hamiltonian is time-independent and named the qubit Hamiltonian $\hat{H}_{qubit}$. The second part is time-dependent $\hat{H}_{int}$, the interaction Hamiltonian. In the following discussion, we will derive an expression for the quantum state at a certain time, which is crucial for subsequently obtaining expressions for various gates that can be applied to the spin qubit. The derivations are based on [45, 46, 43, 47, 44, 48]. Decomposition of the interaction Hamiltonian gives

$$\begin{aligned}\cos(\omega_p t + \phi_p) = \frac{1}{2}\Big[&\big(\cos(\omega_p t + \phi_p)\hat{e}_x + \sin(\omega_p t + \phi_p)\hat{e}_y\big) \\ &+ \big(\cos(\omega_p t + \phi_p)\hat{e}_x - \sin(\omega_p t + \phi_p)\hat{e}_y\big)\Big],\end{aligned} \tag{5.13}$$

which is equivalent to taking two sources. Until now, the Hamiltonians were written in the laboratory (LAB) frame. To continue, we go to the rotating frame, so $\hat{H}_{int}$ is time-independent. Inserting the equation above gives:

$$\begin{aligned}\hat{H}(t) = \frac{\hbar\omega}{2}\hat{\sigma}_z &+ \frac{\hbar\omega_1}{4}\Big[\cos(\omega_p t + \phi_p)\hat{\sigma}_x + \sin(\omega_p t + \phi_p)\hat{\sigma}_y\Big] \\ &+ \frac{\hbar\omega_1}{4}\Big[\cos(\omega_p t + \phi_p)\hat{\sigma}_x - \sin(\omega_p t + \phi_p)\hat{\sigma}_y\Big],\end{aligned} \tag{5.14}$$

where $\Big[\cos(\omega_p t + \phi_p)\hat{\sigma}_x + \sin(\omega_p t + \phi_p)\hat{\sigma}_y\Big]$ and $\Big[\cos(\omega_p t + \phi_p)\hat{\sigma}_x - \sin(\omega_p t + \phi_p)\hat{\sigma}_y\Big]$ are the new frames of the new defined rotating frame. Replace the Pauli operators with ladder operators $\hat{\sigma}_+$ and $\hat{\sigma}_-$, gives some advantages later in the derivations, see Appendix B.1. The first frame can be rewritten as

$$\begin{aligned}\cos(\omega_p t+\phi_p)\hat{\sigma}_x+\sin(\omega_p t+\phi_p)\hat{\sigma}_y &= \begin{pmatrix} 0 & \cos(\omega_p t+\phi_p) \\ \cos(\omega_p t+\phi_p) & 0 \end{pmatrix} \\ &+ \begin{pmatrix} 0 & -i\sin(\omega_p t+\phi_p) \\ i\sin(\omega_p t+\phi_p) & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & e^{-i((\omega_p t+\phi_p))} \\ e^{i((\omega_p t+\phi_p))} & 0 \end{pmatrix} \\ &= e^{i((\omega_p t+\phi_p))}\hat{\sigma}_+ + e^{-i((\omega_p t+\phi_p))}\hat{\sigma}_-, \end{aligned} \tag{5.15}$$

and by analogy, the second frame becomes

$$\cos(\omega_p t+\phi_p)\hat{\sigma}_x-\sin(\omega_p t+\phi_p)\hat{\sigma}_y = e^{-i((\omega_p t+\phi_p))}\hat{\sigma}_+ + e^{i((\omega_p t+\phi_p))}\hat{\sigma}_-. \tag{5.16}$$

Inserting the new frames with ladder operators in the total Hamiltonian gives

$$\begin{aligned}\hat{H}(t) = \frac{\hbar\omega}{2}\hat{\sigma}_z &+ \frac{\hbar\omega_1}{4}\left[e^{i((\omega_p t+\phi_p))}\hat{\sigma}_+ + e^{-i((\omega_p t+\phi_p))}\hat{\sigma}_-\right] \\ &+ \frac{\hbar\omega_1}{4}\left[e^{-i((\omega_p t+\phi_p))}\hat{\sigma}_+ + e^{i((\omega_p t+\phi_p))}\hat{\sigma}_-\right]. \end{aligned} \tag{5.17}$$

Next, a unitary propagator transforms the LAB frame into the rotating frame, where the interaction Hamiltionan is time-independent. The unitary operator is defined as [47, 48]

$$\begin{aligned}\hat{U}(t,0) &= e^{-i\frac{1}{2}\omega_p t\hat{\sigma}_z} = e^{-i\frac{\theta}{2}\hat{\sigma}_z} \\ &= \cos\left(\frac{\theta}{2}\right)\hat{I} - i\sin\left(\frac{\theta}{2}\right)\hat{\sigma}_z, \end{aligned} \tag{5.18}$$

where $\theta = \omega_p t$. The qubit's quantum state in the rotating frame is therefore given by

$$|\Psi_{rot}\rangle = \hat{U}\,|\Psi_{LAB}\rangle\,. \tag{5.19}$$

The transformation of the Hamiltonian in the rotating frame is defined as [45, 48]

$$\hat{H}_{rot} = i\hbar\hat{U}\frac{\partial\hat{U}^{-1}}{\partial t} + \hat{U}\hat{H}_{LAB}\hat{U}^{-1} \tag{5.20}$$

$$= i\hat{\dot{U}}\hat{U}^{\dagger} + \hat{U}\hat{H}_{LAB}\hat{U}^{\dagger}. \tag{5.21}$$

The following steps derive the above equation. First, for unitary $\hat{U}\hat{U}^{\dagger} = \hat{I}$ and Hermitian operators, the inverse operator can be written as

$$\hat{U}^{-1} = \hat{U}^{\dagger} = e^{i\frac{1}{2}\omega_p t\hat{\sigma}_z}. \tag{5.22}$$

Next, the first part of equation 5.20 becomes

$$i\hbar\hat{U}\frac{\partial\hat{U}^{-1}}{\partial t} = ie^{-i\frac{1}{2}\omega_p t\hat{\sigma}_z}\big(i\frac{\omega_p}{2}\hat{\sigma}_z\big)e^{i\frac{1}{2}\omega_p t\hat{\sigma}_z}$$
$$= \frac{-\omega_p}{2}\hat{\sigma}_z \tag{5.23}$$

The second term of equation 5.20 gives

$$\hat{U}\hat{H}_{LAB}\hat{U}^{\dagger} = \frac{\hbar\omega}{2}\hat{U}\hat{\sigma}_z\hat{U}^{\dagger} + \frac{\hbar\omega_1}{4}\Big(e^{i(\omega_p t+\phi_p)}\hat{U}\hat{\sigma}_+\hat{U}^{\dagger} + e^{-i(\omega_p t+\phi_p)}\hat{U}\hat{\sigma}_-\hat{U}^{\dagger}\Big)$$
$$+ \frac{\hbar\omega_1}{4}\Big(e^{-i(\omega_p t+\phi_p)}\hat{U}\hat{\sigma}_+\hat{U}^{\dagger} + e^{i(\omega_p t+\phi_p)}\hat{U}\hat{\sigma}_-\hat{U}^{\dagger}\Big), \tag{5.24}$$

where the ladder operators are sandwiched between the unitary operators. Writing this out gives

$$\begin{aligned}
\hat{U}\hat{\sigma}_+\hat{U}^{\dagger} &= \Big(\cos\big(\frac{\theta}{2}\big)\hat{I} - i\sin\big(\frac{\theta}{2}\big)\hat{\sigma}_z\Big)\hat{\sigma}_+\Big(\cos\big(\frac{\theta}{2}\big)\hat{I} + i\sin\big(\frac{\theta}{2}\big)\hat{\sigma}_z\Big) \\
&= \Big(\cos\big(\frac{\theta}{2}\big)\hat{I} - i\sin\big(\frac{\theta}{2}\big)\hat{\sigma}_z\Big)\Big(\cos\big(\frac{\theta}{2}\big)\hat{\sigma}_+ + i\sin\big(\frac{\theta}{2}\big)\hat{\sigma}_+\hat{\sigma}_z\Big) \\
&= \cos\big(\frac{\theta}{2}\big)^2\hat{\sigma}_+ + i\sin\big(\frac{\theta}{2}\big)\cos\big(\frac{\theta}{2}\big)\hat{\sigma}_+\hat{\sigma}_z + i\sin\big(\frac{\theta}{2}\big)\cos\big(\frac{\theta}{2}\big)\hat{\sigma}_z\hat{\sigma}_+ + \sin\big(\frac{\theta}{2}\big)^2\hat{\sigma}_z\hat{\sigma}_+\hat{\sigma}_z \\
&= \cos\big(\frac{\theta}{2}\big)^2\hat{\sigma}_+ + i\sin\big(\frac{\theta}{2}\big)\cos\big(\frac{\theta}{2}\big)\Big[\hat{\sigma}_+\hat{\sigma}_z - \hat{\sigma}_z\hat{\sigma}_+\Big] + \sin\big(\frac{\theta}{2}\big)^2\hat{\sigma}_+ \\
&= \sin(\theta)\hat{\sigma}_+ + i\cos(\theta)\hat{\sigma}_+ = e^{i\theta}\hat{\sigma}_+,
\end{aligned} \tag{5.25}$$

where we used $[\hat{\sigma}_+, \hat{\sigma}_z] = 2\,\hat{\sigma}_+$, see Appendix B.1, and by analogy

$$\hat{U}\hat{\sigma}_-\hat{U}^{\dagger} = e^{-i\theta}\hat{\sigma}_-. \tag{5.26}$$

Inserting the above in equation 5.24 results in

$$\begin{aligned}\hat{U}\hat{H}_{LAB}\hat{U}^{\dagger} &= \frac{\hbar\omega}{2}\hat{\sigma}_z + \frac{\hbar\omega_1}{4}\Big(e^{i(\omega_p t+\phi_p)}e^{i\omega_p t}\hat{\sigma}_+ + e^{-i(\omega_p t+\phi_p)}e^{-i\omega_p t}\hat{\sigma}_-\Big)\\ &+ \frac{\hbar\omega_1}{4}\Big(e^{-i(\omega_p t+\phi_p)}e^{i\omega_p t}\hat{\sigma}_+ + e^{-i(\omega_p t+\phi_p)}e^{-i\omega_p t}\hat{\sigma}_-\Big)\\ &= \frac{\hbar\omega}{2} + \frac{\hbar\omega_1}{4}\Big(e^{i2\omega_p t}e^{i\phi_p}\hat{\sigma}_+ + e^{-i2\omega_p t}e^{-i\phi}\hat{\sigma}_-\Big) + \frac{\hbar\omega_1}{4}\Big(e^{-i\phi_p}\hat{\sigma}_+ + e^{i\phi}\hat{\sigma}_-\Big)\\ &\cong \frac{\hbar\omega}{2}\hat{\sigma}_z + \frac{\hbar\omega_1}{4}\big(\cos(\phi_p)\hat{\sigma}_x - \sin(\phi_p)\hat{\sigma}_y\big). \end{aligned} \tag{5.27}$$

In the second last line, the terms $e^{i2\omega_p t}$ and $e^{-i2\omega_p t}$ are ignored according to the Rotating Wave Approximation (RWA) [43, 47]. Assuming $\Omega << \omega_p$, the contribution of the terms in the integral in equation 5.9 would be small enough and oscillating so that they can be neglected. Putting it all together:

$$\begin{aligned}\hat{H}_{rot} &= i\dot{\hat{U}}\hat{U}^{\dagger} + \hat{U}\hat{H}_{lab}\hat{U}^{\dagger}\\ &= -\frac{\hbar\omega_p}{2}\hat{\sigma}_z + \frac{\hbar\omega}{2}\hat{\sigma}_z + \frac{\hbar\omega_1}{4}\big(\cos(\phi_p)\hat{\sigma}_x - \sin(\phi_p)\hat{\sigma}_y\big). \end{aligned} \tag{5.28}$$

If we now consider the resonance condition $\omega = \omega_p$ the Hamiltonian becomes

$$\hat{H}_{rot} = \frac{\hbar\omega_1}{4}\big(\cos(\phi_p)\hat{\sigma}_x - \sin(\phi_p)\hat{\sigma}_y\big) \tag{5.29}$$

When $\omega \neq \omega_p$, coherent driving of the qubit would cause a precession movement, resulting in low-quality gates and, hence, a decrease in the fidelity of the quantum channel. The Hamiltonian in the rotating frame can further decomposed into

$$\hat{H}_{rot} = \frac{\hbar\omega_1}{4}\big(\cos(\phi_p)\hat{\sigma}_x - \sin(\phi_p)\hat{\sigma}_y\big) = \frac{\hbar\omega_1}{4}\hat{n}.\hat{\sigma}, \tag{5.30}$$

where $\hat{n} = \big[\cos(\phi_p), -\sin(\phi_p), 0\big]$. Equation 5.9 can further be rewritten in terms of the unitary rotation operator $\hat{U}_{rot}$, see A.2, so that the evolution of a state $|\Psi(t)\rangle$ becomes

$$|\Psi(t)\rangle = \hat{U}_{rot}|\Psi(0)\rangle = e^{-i\frac{\alpha}{4}\hat{n}.\hat{\sigma}}|0\rangle_{rot}. \tag{5.31}$$

Remember that $|\Psi(t)\rangle = \hat{U}|\Psi_{LAB}(t)\rangle = e^{-i\frac{\omega t}{2}\hat{\sigma}_z}|\Psi_{LAB}(t)\rangle$, the initial state in the LAB frame is equal to the initial state in the rotating frame $|0\rangle_{rot} = |0\rangle$ for t = 0. The above equation can be further rewritten as

Table 5.1: Possible combinations of $\Theta(t)$ and ϕ_p values to apply X and Y pulses.

Pulse	$\Theta(t)$	ϕ_p	$\hat{H}_{rot}$
$X_{\pi/2}$	$\pi/2$	0	$\frac{\hbar\alpha}{4}\hat{\sigma}_x$
$Y_{\pi/2}$	$\pi/2$	$\pi/2$	$\frac{\hbar\alpha}{4}\hat{\sigma}_y$
X_{π}	π	0	$\frac{\hbar\alpha}{4}\hat{\sigma}_x$
Y_{π}	π	$\pi/2$	$\frac{\hbar\alpha}{4}\hat{\sigma}_y$

$$\begin{aligned}
|\Psi(t)\rangle &= \left[\cos\left(\frac{\omega_1}{4}t\right)\hat{I} - i\sin\left(\frac{\omega_1}{4}t\right)\hat{n}.\hat{\sigma}\right]|0\rangle \\
&= \left[\cos\left(\frac{\omega_1}{4}t\right)\hat{I} - i\sin\left(\frac{\omega_1}{4}t\right)\left(\cos\left(\phi_p\right)\hat{\sigma}_x - \sin\left(\phi_p\right)\hat{\sigma}_y\right)\right]|0\rangle \\
&= \left[\cos\left(\frac{\omega_1}{4}t\right)\hat{I} - i\sin\left(\frac{\omega_1}{4}t\right)\left(e^{-i\phi_p}\hat{\sigma}_+ + e^{i\phi_p}\hat{\sigma}_-\right)\right]|0\rangle \\
&= \cos\left(\Theta(t)\right)|0\rangle - i\sin\left(\Theta(t)\right)e^{-i\phi_p}|1\rangle\,.
\end{aligned}$$

where $\hat{\sigma}_+|0\rangle = |1\rangle$, $\hat{\sigma}_-|0\rangle = 0$ and $\Theta(t) = \frac{\omega_1}{4}t$. The above equation provides a detailed positioning of the qubit's quantum state, in function of the polar angle $\Theta(t)$ and azimuthal angle ϕ_p. So, coherent driving during a certain time brings the qubit into superposition or induces a spin flip. And through equation 5.30, $\phi_p = 0$, gives a rotation around the x-axis while $\phi_p = \pi/2$ results in a rotation around the y-axis. Changing the phase of the RF source changes the qubit's rotation axis. With this information, quantum gates performing X- and Y-pulses are gathered in table 5.1.

Figure 5.6 shows an RF pulse to drive the qubit. The execution time of the RF pulse results regulates the polar angle of the qubit quantum state. Changing the phase of the RF pulse pushes the qubit to another rotation axis. When driving the qubit, the probability of measuring the spin-up fraction will change according to the polar angle $\Theta(t)$ in function of time. Calculating the spin-up probability gives

$$\begin{aligned}
P\Big(|1\rangle\Big) &= |\langle 1|\,|\Psi(t)\rangle\, e^{-i\phi_p}|^2 \\
&= |-i\sin\left(\Theta(t)\right)|^2 \\
&= \sin\left(\Theta(t)\right)^2
\end{aligned} \tag{5.32}$$

and is depicted in figure 5.32. The oscillating probability of the spin-up fraction is also known as Rabi-oscillations.

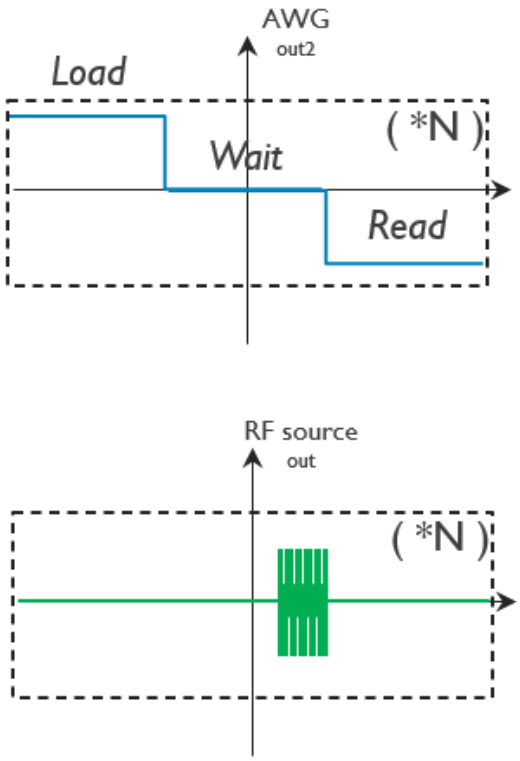


Figure 5.6: Above, a block wave voltage source (plunger voltage V_p) on the quantum dot regulates the load and read steps. The manipulation occurs during the waiting time when the RF pulse is applied. AWG stands for Arbitrary Wave Generator, the voltage source. Under, a graphic representation of the RF pulse during the waiting time and just before the qubit readout.

In an ideal world, the oscillation would continue for an infinite amount of time. But due to the decoherence of the qubit, the Rabi-oscillations are damped according to the Rabi decay time $T_{2,Rabi}$:

$$P(|1\rangle) = e^{-\frac{t}{T_{2,Rabi}}}. \tag{5.33}$$

The oscillation amplitude decreases exponentially. As mentioned earlier, an RB experiment requires a proper decoherence time to execute the full quantum gate sequence. Starting from a gate-controlled quantum dot, we now achieved coherence driving of a qubit in the quantum dot. The following sections tend to apply quantum gates on a working qubit to perform an RB experiment. But first, the qubit needs to be calibrated.

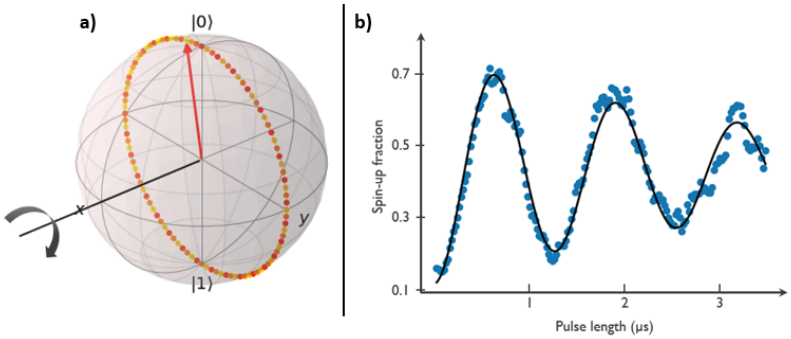


Figure 5.7: a) Bloch sphere representation of coherent driving of the qubit around the x-axis. b) Rabi-oscillations (spin-up fraction vs pulse length), according to equation 5.32.

5.4 Calibration

The calibration is twofold: first, the qubit's resonance frequency (ω from the previous section) needs to be calibrated. Coherent driving with a frequency other than the resonant frequency results in a precession movement of the qubit in the Bloch sphere, hence, a vast decrease in fidelity. Although the resonance frequency is not a parameter needed for applying the gates or measuring the spin-up fraction, it is an important parameter in the experimental setup. That will come in the next sections. Measuring the spin-up fraction while stepping through the driving frequency results in a Gaussian shape graph. The spin-up fraction can be fitted with Gaussian given by

$$f(x) = A \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{5.34}$$

where μ is the position of the center of the peak (the average of the distribution), σ the standard deviation, and A the amplitude. Hence, the resonance frequency is denoted by μ the center of the peak. The deviation of the frequency ($\delta\omega$) is defined through the full-width-at-half-maximum (FWHM)

$$\delta\omega = 2\sqrt{2\ln 2}\sigma. \tag{5.35}$$

The resonance frequency is typically in the orders of 100 kHz - 10 MHz. Another important parameter that determines the quality of the experimental setup is the visibility, defined as the difference between the maximum and minimum measured spin-up fraction. The visibility is limited by the read-out fidelity of the qubit and the duration of the pulse T_{tot} compared with the T_1 time: $T_{tot} << T_1$. A qubit in a performant experimental setup has a visibility in the range of [max, min] = [0.1, 0.8].

The second calibration needs to be performed to extract the duration of the π and $\pi/2$-pulses. Referring to Figure 5.7, the measurement of Rabi-oscillations provides the period (T_p) of the qubit to do exactly one full rotation around the Bloch sphere. The π and $\pi/2$-pulses are evidently given through $T_p/2$ and, $T_p/4$ respectively. The same resonance frequency applies to rotation around the x- and y-axis.

5.5 Practical implementation of the RB protocol

The practical implementation of an RB protocol consists of interleaving the so-called computational gates and Pauli gates, the method was first described by E. Knill [12] and later by [49] and [50]. First, the qubit is prepared in a known initial quantum state ρ [1]. Next, an alternating sequence of either π-pulses and $\pi/2$-pulses or identity operators is executed on the qubit, followed by a reverse gate to bring the qubit back into the initial state. At last, a measurement provides the spin-up fraction. The single-qubit RB protocol is a randomization of the 48 operations which can be parameterized as

$$\begin{aligned} \mathcal{SP} &= \exp\left(\pm i\frac{\pi}{4}\mathcal{Q}\right)\exp\left(\pm i\frac{\pi}{2}\mathcal{V}\right), \\ \mathcal{Q} &\in \{\sigma_x, \sigma_y, \sigma_z\}, \mathcal{V} \in \{\mathbf{1}, \sigma_x, \sigma_y, \sigma_z\} \end{aligned} \tag{5.36}$$

where $\mathcal{S}$ is are the computational gates and $\mathcal{P}$ the Pauli gates. A graphical representation of the sequences is shown in Figure 5.8. The $\pi/2$-pulses are supposed to advance computation and are therefore referred to as computational gates or pulses. The π-pulses randomize the error, it can be seen as a change in the Pauli frame. The randomization of the pulse sequence is important for the RB protocol, randomization ensures the outcome is not correlated with any individual pulse or proper subsequence of pulses [12].

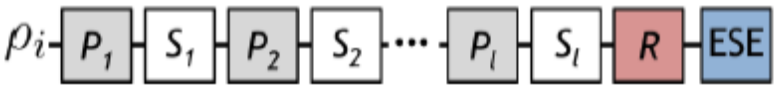


Figure 5.8: RB sequence, interleaved Pauli gates (π-pulses) and computational gates ($\pi/2$-pulses). At the end, the reverse pulse and the measurement.
Source: Park, D. et al. [50]

[1] The initial quantum state is defined by one measure-read-out step. It can be done in two ways: The first method is to Load and Read. If the measurement is 0, a spin down $|0\rangle$ (ground state spin) is in the quantum dot. If the measurement is not zero, the electron is replaced by an electron in the ground state. The second method is to wait (relaxation time), so the spin particle is in the spin-down ground state.

The standard RB protocol is contained in the Python class *OneQ_SRB_experiment.py* and can be consulted in Appendix D.3. The class generates the sequences and executes them on a quantum system through a measurement file containing the specific device parameters and commands. To initiate the standard RB protocol, the following parameters need to be specified:

- **sequence lengths**
  A list of integers: sets the sequence lengths, e.g., $[5, 10, 25, 50, 100]$.
- **sample amount**
  Integer: sets the amount of sequences to be generated for each sequence length.
- **seeds**
  A list of integers: to define a seed for each sequence individually. This parameter can be used to acquire the preferred randomization and each experiment can be exactly repeated with the same seed list. The sampling method described by [12] consists of random generation of computational gates sequences, which are truncated in random lengths and interleaved by random generated Pauli gate sequences of the same length. This method provides a generic sampling of sequences which are all random and different from each other. However, because we define a different seed for each sequence, all sequences are randomly generated and different from each other, so the Monte-Carlo simulation still holds.
- **shots**
  Integer: defines the number of times each sequence needs to be executed, and is the execution of the load-wait-read step. The default value is 100 times, which repeats 100 times the load-wait-read step.
- **pulses**
  Dictionary $\{$"π_pulse" : $T_p/2$, "$\pi/2\_pulse$" : $T_p/4\}$: contains the calibrated $T_p/2$ and $T_p/4$ time for the respectively π and $\pi/2$ pulses, with rotation around the x- and y-axis. The periods need to be specified in terms of the clock time, e.g., $T_p/2 = 1000$ means the time needed to execute a π-pulse is 1000 times the clock time.
- **clock time**
  Sets the sample rate of the arbitrary wave generator. the clock time is the resolution at which the equipment can sample. The spin-up fraction can usually be measured every nanosecond.

Analog to the Qiskit simulations, the function *transpiler* transpiled a given gate into a pulse represented by a dictionary. The dictionary contains all the information about the pulse needed to execute the standard RB protocol: the rotation axis, pulse angle and pulse time. The pulse angle is the polar angle (in units of π), which keeps track of the position of the qubit with respect to the z-axis. The pulse time is the

Table 5.2: Preprogrammed gates in the transpiler. P stands for Pauli gates or π-pulses. S stands for symplectic or computational gates ($\pi/2$-pulses). The -X and -Y rotation axis is a rotation around the X/Y axis with a phase shift over π. Pulse time is in units of the clock time.

Gate	Rot. axis	Pulse angle	Pulse Time
P_1	X	π	$T_p/2$
P_2	Y	π	$T_p/2$
S_1	X	$\pi/2$	$T_p/4$
S_2	-X	$3\pi/2$	$T_p/4$
S_3	Y	$\pi/2$	$T_p/4$
S_4	-Y	$3\pi/2$	$T_p/4$

$T_p/2$ or $T_p/4$ time. The gates in the *transpiler* function are gathered in table 5.2. We choose identity gates to be a combination of two computational gates ($-\pi/2$ and $\pi/2$), see figure 5.9. The I_1 identity gate is a combination of the S_2 and S_1 gates ($3\pi/2 + \pi/2$ around X). And the I_2 identity gate is a combination of the S_4 and S_3 gates. The transpiler contains also some combinations of pulses to generate Hadamard gates according to the RB protocol depicted in [4, 7] for further research.

The function *pulse_sequence* builds up the sequences according to Figure 5.8, using the seeds to select gates randomly, and transpiles them. The reverse gate is determined through the sum of the polar angles of the pulses θ_T, and the following final angle θ_f estimation:

$$n = floor\left(\frac{\theta_T}{2}\right) \tag{5.37}$$

$$\theta_f = \theta_T - 2n. \tag{5.38}$$

If $\theta_f = 0$, the I_1 identity gate is returned. S_1 if $\theta_f = 3\pi/2$ and S_1 for $\theta_f = \pi/2$. The P_1 gate for $\theta_f = \pi$. The most difficult part is knowing the phase shift after each gate. The phase shifts are fundamental for the computational gates and is essential for making quantum gates and make quantum computers think. The function *phase_shift* decides which phase shift needs to be assigned at each gate. If the new axis is equal to the current axis, the current axis is returned and no phase

Table 5.3: Phase shift ϕ_p for the new axis for each new gate. The top half is for a polar angle $\theta < \pi$ and the bottom half is for a polar angle $\theta > \pi$. The axes $-X$ and $-Y$ indicates an extra phase over π for the $S_{2,4}$ gates. The phase stays unchanged if the new and current axis are the same.

current/new axis	X	Y	-X	-Y
X	/	$\pi/2$	π	$-\pi/2$
Y	0	/	π	$-\pi/2$
X	/	$-\pi/2$	0	$\pi/2$
Y	π	/	0	$\pi/2$

shift is visible on the RF pulse. If the rotation axis changes or a $3\pi/2$-pulse (-X, -Y) is given, the function uses of the phase matrices in table 5.3, to obtain the correct phase shift. For example, if the quantum state is in superposition, in the xy-plane (with polar angle $\theta = \pi/2$), and rotating around the Y axis. The phase shift needed for an S_2 gate on that moment is $\phi_p = \pi$, so the quantum state ends up in the initial $|0\rangle$ state with a rotation around the x-axis. A last exception needs to be taken into account when the new phase, given through the phase matrix, is the same as the current phase and a $S_{2,4}$ gate is given. In those special case an extra π phase shift needs to be added.

Eventually, the functions *sequence_sampler* and *build_randomized_benchmarking* samples the sequences and build up the RB protocol. The result is a dictionary containing all the generated sequences and a second dictionary containing some information about each sequence, total pulse time, total polar angle, and phase shift and used seed. An example can be consulted in Appendix C.2.

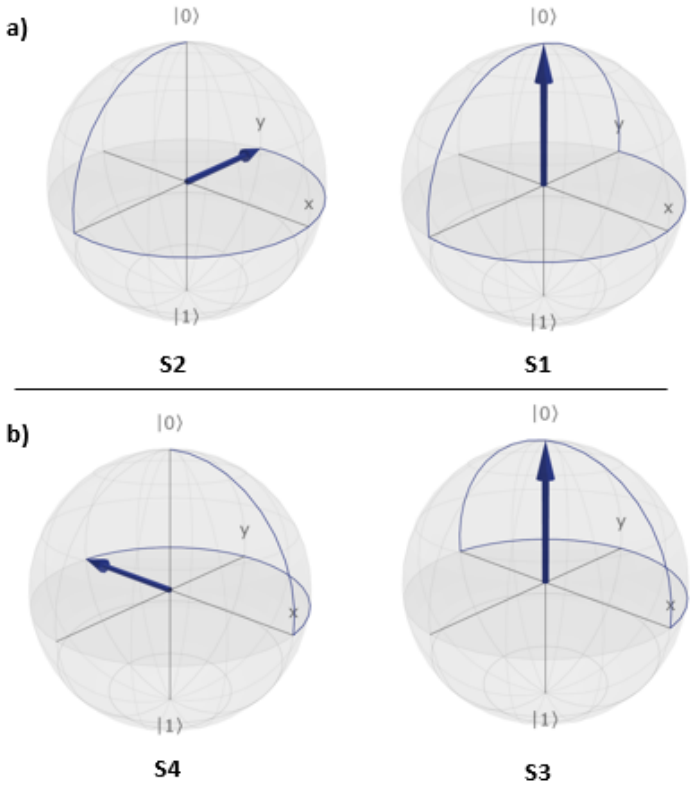


Figure 5.9: Decomposition of the identity gates in computational (S) gates. a) I_1 is a sequence of the S_2 and S_1 gate around the x axis. b) I_2 is a sequence of the S_4 and S_3 gate around the y axis. Figures adapted from: @ Konstantin Herb — ETH Zurich (Bloch sphere visualizer)

5.6 Experimental set-up

To operate the qubit, the quantum dot needs to be conserved in a cryogenic environment so that $k_B T_e << \hbar\omega$, where T_e is the electron temperature and ω the qubit resonance frequency. An increase in thermal energy results in a loss of coherence due to the spin-phonon coupling and, in general, loss of qubit control. Therefore, a quantum computer can be seen as a fridge at cryogenic temperatures ($\sim$ 10mk). A picture of the inside of the fridge is shown in Figure 5.10. Liquid helium is inserted at the top of the fridge. In the 4K region, liquid helium is a mixture of isotopes ^{4}He and ^{3}He. When cooled below 0,87K [51] (almost at the bottom of the fridge), liquid helium undergoes spontaneous phase separation, forming a layer ^{3}He-rich phase ^{3}He (+ ^{4}He) and underneath a more dense ^{3}He-poor phase ^{4}He (+ ^{3}He) layer. The ^{3}He in the ^{3}He-poor phase evaporates, causing heat to be extracted from the system. In the 4K region, ^{3}He vapor is compressed and condensates back into the liquid phase.

A pump forces the liquid phase back down to complete the cycle. At the bottom of the fridge, temperatures around 10mK can be reached, hence, the quantum dot is mounted in the 10mK region.

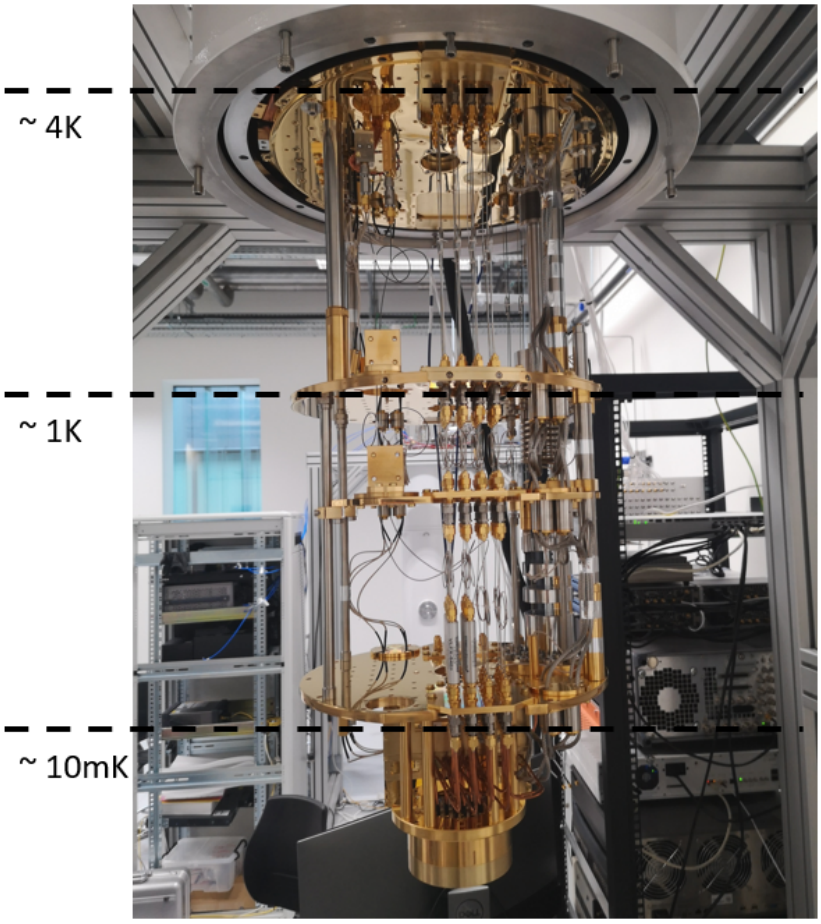


Figure 5.10: Quantum computer (inside), the temperature regions are indicated with a black dashed line.

Aside from a fridge at cryogenic temperatures, several devices, such as an Arbitrary Wave Generator (AWG) with a Performance Signal Generator (PSG), are needed for qubit control and a digitizer for sampling the signal waveforms. A schematic overview of the cryogenic measurement setup is depicted in Figure 5.11. The first channel, applied to the qubit's plunger gate V_p, is for the spin read-out. The other two input channels of the AWG are kept for IQ modulation, and a fourth for the digitizer. The execution of a pulse sequence can be summarized as follows: during the load period of the quantum dot, a pulse drives the qubit, where several

phase shifts are performed through IQ mixing. This is the step where the quantum gates are executed on the qubit. After a short waiting time, as shown in Figure 5.11, the trigger measures the current to obtain the spin-up fraction, the read step. If the RB protocol measured each sequence k times (the number of shots), the load-read sequence is repeated k times.

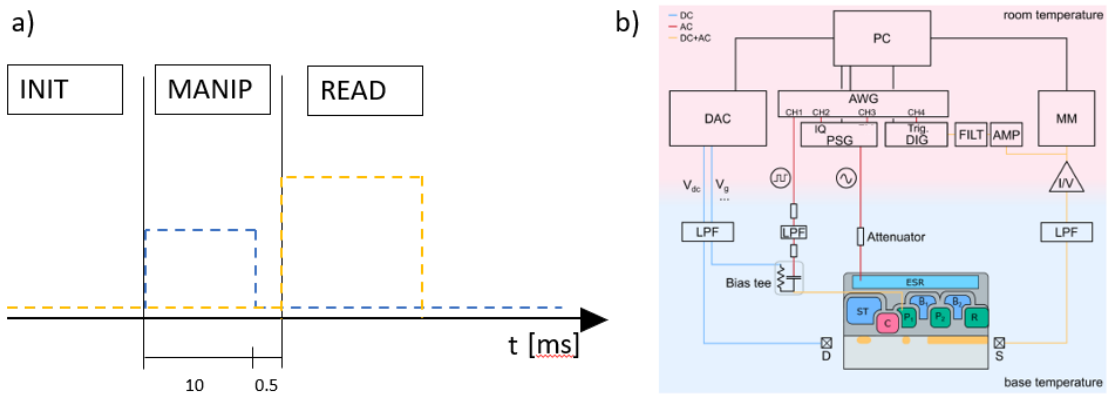


Figure 5.11: A) The initialization-manipulation-read sequence, the blue block curve, forms a time window to execute the RF pulse during the spin manipulation. The yellow curve reads the spin-up fraction in the dot. b) The first channel is for the spin readout, and the second and third are for the IQ modulation. The equipment in the red area is controlled at room temperature. The cryogenic temperatures apply in the blue zone. Figure b) is adapted from Dumoulin Stuyck, N. (KUL) [41].

5.6.1 IQ-mixer

IQ-mixing allows us to create a phase shift during the pulse and is the key procedure for generating quantum gates on a spin qubit system. IQ-mixers use quadrature modulation to maximize the information transmission in a limited bandwidth and are therefore widely used in telecommunications [52]. Quadrature modulation allows precise control of the amplitude and phase of the output signal through modulating both the in-phase and quadrature components of a signal. IQ modulation can be understood by observing Figure 5.12. According to [52] and [53], an IQ mixer consists of two regular mixers and a quadrature hybrid coupler in the local oscillator (LO). The signal of the LO is split by the hybrid coupler with a 90° phase shift into two output ports. We distinguish two cases: upconversion and downconversion. Upconversion means the outputs are mixed with the In-phase signal or the Quadrature signal, and then both outputs are combined in the RF port. The inverse result is achieved with downconversion. Both I and Q signals can be retrieved out of the RF input signal. Only upconversion is used to generate the qubit pulse.

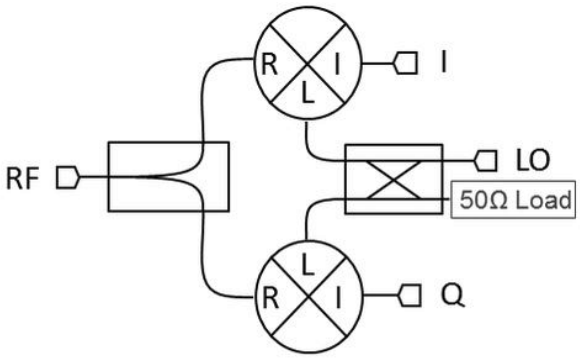


Figure 5.12: Internal structure of an IQ mixer, with the I and Q channels on top and bottom and the local oscillator in the middle. Source: Abadal, A. (ETH) [52]

More precisely, a single sideband upconversion generates the qubit pulse. Due to the imperfections of the mixers, a certain leakage of the LO signal will go to the RF port. Hence, there is still a certain drive at the qubit frequency even if no voltage is applied to the I and Q ports. For this reason, most IQ mixers operate with oscillating signals in the I and Q ports. The waveforms are given by [35]

$$s_I(t) = I\cos\left(\omega_{IF}t + \phi\right) \tag{5.39}$$
$$s_Q(t) = Q\sin\left(\omega_{IF}t + \phi\right), \tag{5.40}$$

where ϕ is the phase shift and ω_{IF} is the intermediate frequency, i.e., the frequency between the two peaks of the local oscillator and the upconversion in the power spectrum. $s_I(t)$ and $s_Q(t)$ are voltage signals generated with the AWG (with a sampling rate of 1/clock time). When performing single-sideband mixing, we assume I = Q = A. The resulting radiofrequency pulse equals

$$A\cos\left(\omega_{IF}t + \phi\right)\cos\left(\omega_{LO}t\right) \mp A\sin\left(\omega_{IF}t + \phi\right)\sin\left(\omega_{LO}t + \phi\right) = A\cos\Big(\left(\omega_{IF} \pm \omega_{LO}\right)t + \phi\Big), \tag{5.41}$$

where ω_{LO} is the local oscillator frequency. The RF signal is twofold, with frequencies $\omega_{LO} + \omega_{IF}$ for upconversion and $\omega_{LO} - \omega_{IF}$ for downconversion. In upconversion, if the qubit resonance frequency equals $\omega = \omega_{LO} + \omega_{IF}$, the ESR technique performs coherent driving on the qubit to apply quantum gates. The rotation axis can be manipulated by changing the phase ϕ in the IQ mixer. The Keysight AWG [54] has a built-in PSG signal generator to generate the LO signal. The intermediate frequency must satisfy the following boundaries:

$$\Delta f < \omega_{IF} < BW, \tag{5.42}$$

where $\Delta_f \sim 1$ MHz is the FMHW from the Gaussian fit in the calibration. BW stands for the AWG's bandwidth ($BW \sim 100$ MHz) and the qubit resonance frequency $\omega_{LO} + \omega_{IF} \sim 10$ GHz.

In the Class *OneQ_SRB_experiment*, the function *build_randomized_benchmarking* (see Appendix D.3) generates all the sequences to execute an RB protocol. An example of general results and sequences is given in Appendices C.2 and C.3. Next, the function *build_arrays* returns the time and phase list to build up the sine and cosine signals in the AWG. Some examples of sequences executed through the AWG on the digitizer are depicted in figures 5.13, 5.14, 5.15 and 5.16. The first two figures show a sequence of three pulses. The first sequence contains the P2 (π), S3 ($\pi/2$), and S1 ($\pi/2$) gate, with one phase shift to go from a rotation around the y-axis to a rotation around the x-axis. The sequence in figure 5.14 starts with a y rotation over π (P2 gate) to go to $3\pi/2$ rotation around the x-axis. Followed by another $3\pi/2$ rotation around the x-axis, which makes two phase shifts: the first one to change the rotation axis (from y to x) and perform a π shift, the second one to perform again a π shift. Because of the two $3\pi/2$ pulses (S2 gates) in sequence 3. The same holds for the other random generated sequences, the last two figures. Sequence 8, depicted in figure 5.15, contains five gates and 6 pulses and performs a total polar angle of 4π. The last example, sequence 12, contains seven pulses and performs a total polar angle of 6π. The pulse, as shown through the digitizer, contains four phase shifts so that the qubit final state equals the initial state. The initial state is chosen to be the spin-up state in block sphere representation. The code works for any initial state and initial rotation axis since the final polar angle must satisfy an even number of π, and the (ϕ=0)-axis is set at the beginning of each experiment. Although we cannot execute a real RB experiment yet on the quantum device at Imec, these pulses generated by the AWG and sent to the digitizer proof that the protocol can be executed on their experimental setup. When the staff in the research group deems it necessary.

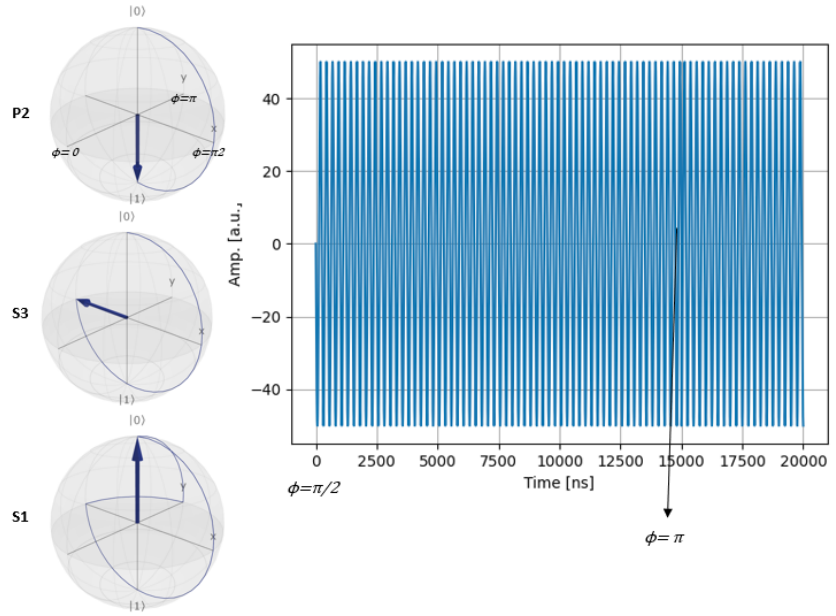


Figure 5.13: Pulse of sequence 1: 'rot axis': ['Y', 'Y', 'X'], 'pulse time': [10000, 5000, 5000], 'phase shift': [0.5, 0.5, 1.0], 'gates': ['P2', 'S3', 'S1']

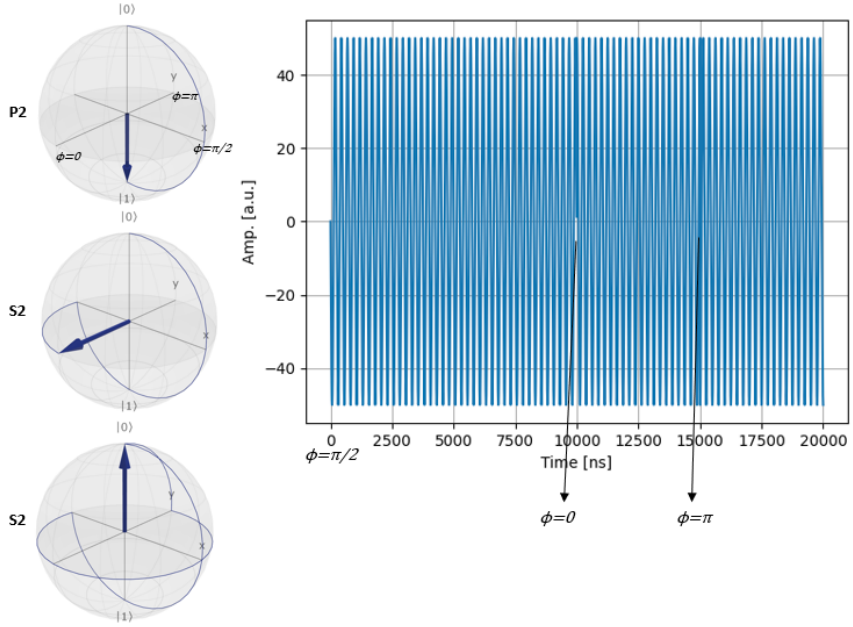


Figure 5.14: Pulse of sequence 3: 'rot axis': ['Y', '-X', '-X'], 'pulse time': [10000, 5000, 5000], 'phase shift': [0.5, 0.0, 1.0], 'gates': ['P2', 'S2', 'S2']

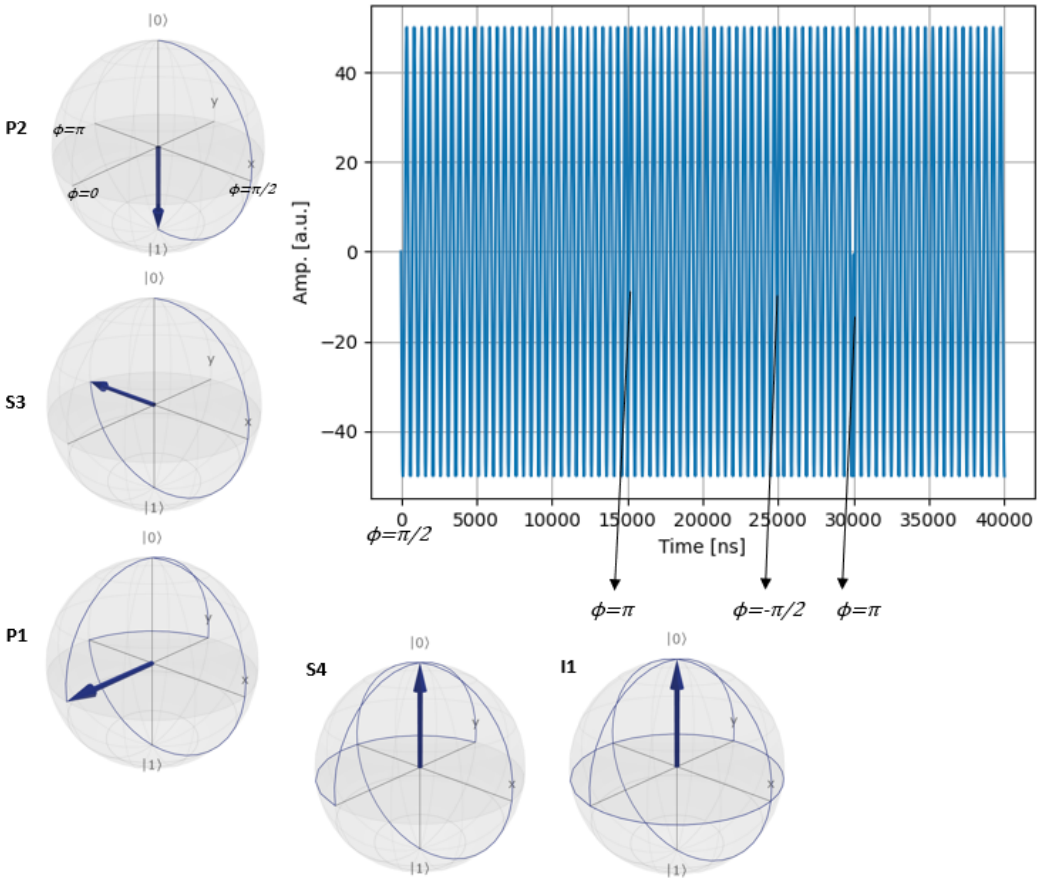


Figure 5.15: Pulse of sequence 8: 'rot axis': ['X', '-X', '-X', 'X', '-Y', '-X', 'X', '-X', 'X'], 'pulse time': [10000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000], 'phase shift': [0.0, 1.0, 0.0, 0.0, -0.5, 1.0, 1.0, 0.0, 0.0], 'gates': ['P1', 'S2', 'I1', 'S4', 'I1', 'S2', 'S1']

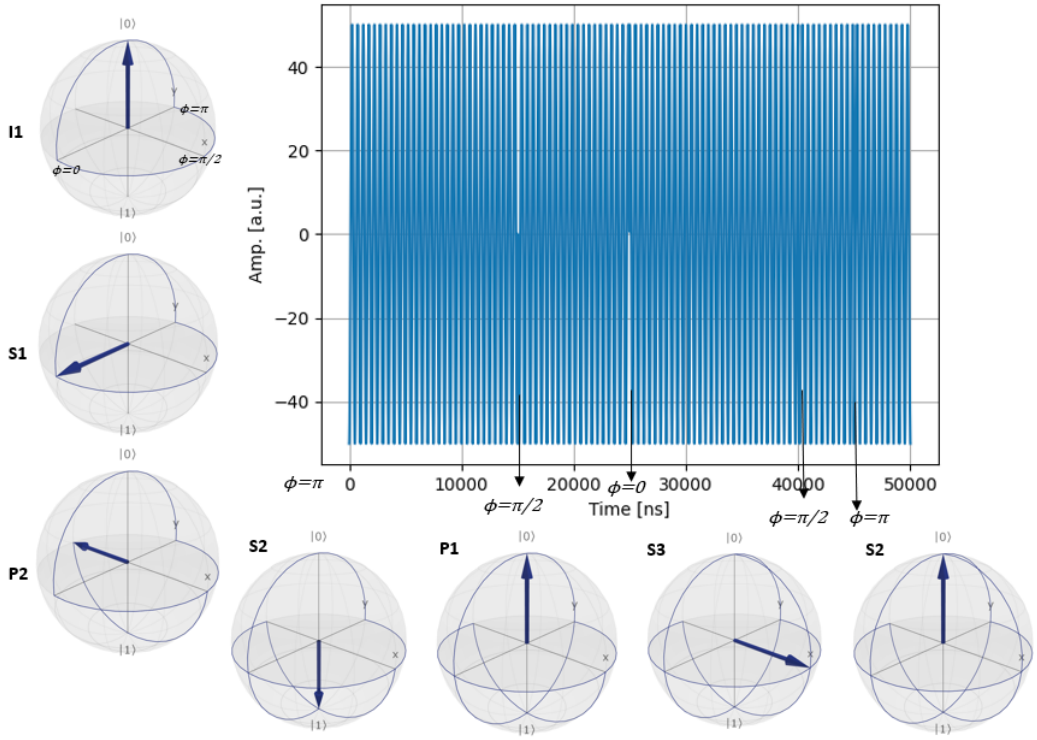


Figure 5.16: Pulse of sequence 12: 'rot axis': ['-X', 'X', 'X', 'Y', '-X', 'X', 'Y', '-X'], 'pulse time': [5000, 5000, 5000, 10000, 5000, 10000, 5000, 5000], 'phase shift': [1.0, 1.0, 1.0, 0.5, 0.0, 0.0, 0.5, 1.0], 'gates': ['I1', 'S1', 'P2', 'S2', 'P1', 'S3', 'S2']

Chapter 6

Conclusion

Randomized Benchmarking is an efficient and robust method that is widely used in practice to estimate the average fidelity of a gate set implemented on a quantum computing device. Benchmarking a quantum computer is a scalable process that is robust to SPAM errors. The thesis discussed Standard Randomized Benchmarking, the most hands-on method to extract the average fidelity of a quantum system, and Interleaved Randomized Benchmarking to retrieve the characteristics of a single gate on a quantum device. The advantage of using an RB protocol is the ability to isolate SPAM errors and simultaneously characterize both SPAM errors and gate errors. The price paid is that only the average gate fidelity can be extracted by amplifying the gate errors in a depolarizing channel while leaving the SPAM errors constant. Hence, the A_0 and B_0 parameters indicate the influence of SPAM errors or the quality of the experimental setup, which is one of the major advantages of RB. The mathematical insights to derive an RB protocol can be summarized as follows:

1. If a quantum channel behaves as a depolarizing channel, the average gate fidelity can be extracted:

$$\overline{F}_g = p + \frac{1-p}{d}.$$

2. The twirled operation of a unitary 2-design, such as the Clifford group, is a depolarizing channel:

$$\frac{1}{K}\sum_j C_j^\dagger \Lambda C_j \approx \Lambda(\rho) = p\rho + (1-p)\frac{I}{d}.$$

3. Gate and time-independent errors lead to an exponential decay of the fidelity:

$$F_{seq}^{(0)}(m,\psi) = A_0 p^m + B_0 \tag{6.1}$$

where the depolarizing parameter p indicates the probability of staying in the initial state. To gain some experience with RB experiments, the simulations in

Chapter 4 form a good basis for ultimately moving to quantum computers in real experimental setups. A physical qubit is obtained by isolating an electron in a quantum dot and apply a magnetic field. The Zeeman splitting splits the degeneracy of the energy of an isolated electron in a $|0\rangle$- or $|1\rangle$-state. Comparable with the 0 or 1 values in a classical bit. The electron spin resonance technique enables the coherent manipulation of the spin state. The qubit starts to rotate from the spin-up state to the spin-down state and back. The rotation axis can be shifted by changing the phase of the applied radiofrequency pulse. IQ mixing can experimentally regulate the phase shift where a sine and cosine wave are generated through an AWG performing single-sideband upconversion. The output is a cosine radiofrequency pulse, where the phase of the qubit can be changed through the phase of the cosine wave. Eventually, a Python Class generates sequences to execute RB experiments on the experimental set-up at Imec. We were able to generate the random sequences on the digitizer, which proves that the experimental implementation of the RB protocol, as described and executed in this thesis, can be used for further research purposes. Although we cannot yet execute a real RB experiment on the quantum device at Imec, with this preliminary research, the researchers at Imec are hopefully already a step closer to developing high-performance quantum devices with acceptable accuracy.

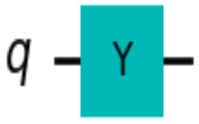
6.1 outlook

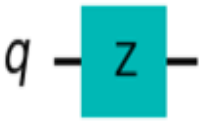
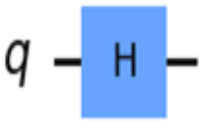
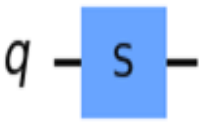
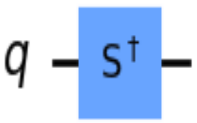
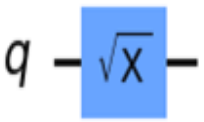
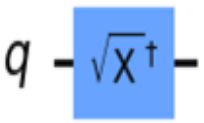
So what's next? The question arises of how the math changes when performing an RB experiment on two-qubit devices or more. It seems that the mathematical formulation of the averaging of the quantum channel (The twirl) changes in multi-qubit devices [55]. Researchers from the Technical University of Delft (TU Delft) merely use character randomized benchmarking for two-qubit devices [8, 7, 9], where several RB experiments are executed. When the first or the second qubit is idle, and when there are two-qubit gates on both qubits. Character randomized benchmarking is already widely used for two-qubit systems, but what about three-qubit systems and more? The coupling map of two qubits is always linear, but from three qubits or higher, multiple coupling maps can be considered for the same multi-qubit device; how will the Twirl change? And can we still average over coupling maps with complex structures? Besides the coupling maps, the size of the Clifford scales with $2^{O(n^2)}$ [33]. So the gate set of Clifford gates becomes very large very quickly, so it's difficult to uniformly sample the gate set to average over the quantum channels. Which is necessary for our Twirl and extracting the depolarizing parameter. There are still many unanswered questions, which makes RB a very interesting topic in the world of quantum computers. The future will show whether RB is the best method to determine the accuracy of many qubit systems. In a world where the increasing computational power of computers is becoming increasingly important, good and scalable accuracy determination is important for the progress of many scientific research.

Appendix A

Common quantum gates

A.1 Single qubit Clifford gates

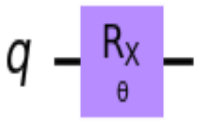
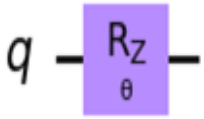
Type (symbol)	Matrix	Diagram
I	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	
X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	
Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	

Type (symbol)	Matrix	Diagram
Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	
H	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	
S	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	
Sdg	$\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$	
SX	$\frac{1}{2}\begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$	
SXdg	$\frac{1}{2}\begin{pmatrix} 1-i & 1+i \\ 1+i & 1-i \end{pmatrix}$	

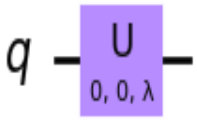
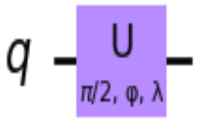
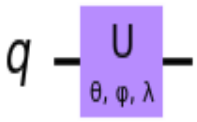
A.2 Single qubit rotation gates

General expression:

$$R_{\hat{n}}(\theta) = \exp\left\{-i\frac{\theta}{2}\hat{n}.\hat{\sigma}\right\} = \cos\left(\theta/2\right)I - i\sin\left(\theta/2\right)\left(n_x X + n_y Y + n_z Z\right) \qquad \text{(A.1)}$$

Type (symbol)	Exp. repr.	Matrix	Diagram
$\mathbf{RX}(\theta)$	$\exp\{-i\frac{\theta}{2}X\}$	$\begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$	
$\mathbf{RY}(\theta)$	$\exp\{-i\frac{\theta}{2}Y\}$	$\begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$	
$\mathbf{RZ}(\theta)$	$\exp\{-i\frac{\theta}{2}Z\}$	$\begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$	

A.3 Unitary gates U

Type (symbol)	Matrix	Diagram
U1(0, 0, λ)	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$	
U2($\pi/2, \varphi, \lambda$)	$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\varphi} & e^{i(\varphi+\lambda)} \end{pmatrix}$	
U3(θ, φ, λ)	$\begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\varphi}\sin(\theta/2) & e^{i(\varphi+\lambda)}\cos(\theta/2) \end{pmatrix}$	

Appendix B

Ladder operators $\hat{\sigma}_+$ and $\hat{\sigma}_-$

B.1 single qubit ladder operators

The $\hat{\sigma}_+$ operator is defined as

$$\hat{\sigma}_+ = \frac{\hat{\sigma}_x - i\hat{\sigma}_y}{2} = \frac{1}{2}\left[\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} - i\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}\right] = \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix}. \tag{B.1}$$

The can be used to hop from the ground state to the excited state:

$$\hat{\sigma}_+|0\rangle = \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \begin{pmatrix}0\\ 1\end{pmatrix} = |1\rangle. \tag{B.2}$$

The $\hat{\sigma}_-$ operator is defined as

$$\hat{\sigma}_- = \frac{\hat{\sigma}_x + i\hat{\sigma}_y}{2} = \frac{1}{2}\left[\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} + i\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}\right] = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}, \tag{B.3}$$

and can be used to hop from the excited state to the ground state:

$$\hat{\sigma}_-|1\rangle = \begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix} = \begin{pmatrix}1\\ 0\end{pmatrix} = |0\rangle \tag{B.4}$$

.

The commutator of $\hat{\sigma}_+$ and $\hat{\sigma}_z$ equals

$$\left[\hat{\sigma}_+, \hat{\sigma}_z\right] = \hat{\sigma}_+\hat{\sigma}_z - \hat{\sigma}_z\hat{\sigma}_+ = \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} - \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix} \tag{B.5}$$

$$= \begin{pmatrix}0 & 0\\ 1 & 0\end{pmatrix} - \begin{pmatrix}0 & 0\\ -1 & 0\end{pmatrix} = 2\hat{\sigma}_+. \tag{B.6}$$

Appendix C

Generated data

C.1 Simulation: Aer noise model

Parameters:
depolarizing parameter p = 0,01 on the U2 gate and a thermal relaxation error depicted in appendix A.3 on the U3 gate.
lengths = [5, 10, 25, 50, 100, 150, 200, 250, 300, 350]
samples = 10
seeds = list(range(100))

asked_length	QC_length	TQC_length	used_seed	th_fidelity	state_fidelity
5	11	11	0	0.9646	0.9615620560100206
5	11	11	1	0.9632	0.9615117096829614
5	11	11	2	0.9611	0.9615155971927111
5	11	11	3	0.9621	0.9615620560100211
5	11	11	4	0.988	0.989801387085105
5	11	11	5	0.9748	0.9755717692100837
5	11	11	6	0.9555	0.95691959144491
5	11	11	7	0.9759	0.9756191008655037
5	11	11	8	0.9859	0.985129253893006
5	11	11	9	0.9594	0.9615117096829614
10	21	21	10	0.9837	0.9848349097235088
10	21	21	11	0.9407	0.9431080528722993
10	21	21	12	0.9945	0.9941241384010402
10	21	21	13	0.9834	0.9849317947529997
10	21	21	14	0.9495	0.9473650825934744
10	21	21	15	0.9488	0.9521018690358476

asked_length	QC_length	TQC_length	used_seed	th_fidelity	state_fidelity
10	21	21	16	0.9842	0.98458983922714
10	21	21	17	0.9299	0.9303247431539801
10	21	21	18	0.9281	0.9260290032571851
10	21	21	19	0.9322	0.934372020970836
25	51	51	20	0.8604	0.8664107500290766
25	51	51	21	0.9231	0.925298139367773
25	51	51	22	0.9015	0.9006105532358781
25	51	51	23	0.8747	0.8728453268127124
25	51	51	24	0.8525	0.8518837635815769
25	51	51	25	0.8456	0.8487408831898845
25	51	51	26	0.8936	0.8921350739521197
25	51	51	27	0.8692	0.8659952756155664
25	51	51	28	0.9049	0.9012033940200237
25	51	51	29	0.8436	0.8517169834442525
50	101	101	30	0.7594	0.7650956659238395
50	101	101	31	0.8127	0.8174578453512142
50	101	101	32	0.7699	0.7660929945346199
50	101	101	33	0.7195	0.7177418797558541
50	101	101	34	0.7828	0.7818252602865665
50	101	101	35	0.7672	0.7704232497329444
50	101	101	36	0.7496	0.7499515326968702
50	101	101	37	0.8658	0.863262119315276
50	101	101	38	0.7646	0.7599319180921381
50	101	101	39	0.7856	0.7788689750779152
100	201	201	40	0.6323	0.626007712493456
100	201	201	41	0.6343	0.6386068998259388
100	201	201	42	0.645	0.6381588039139406
100	201	201	43	0.6342	0.6339671670429015
100	201	201	44	0.6442	0.6441291501144708
100	201	201	45	0.6619	0.6649255347506557
100	201	201	46	0.6281	0.624901933618588
100	201	201	47	0.7	0.6970382623773916
100	201	201	48	0.6358	0.6302673886098452
100	201	201	49	0.6717	0.6648042570966017
150	301	301	50	0.5698	0.5795573898347246
150	301	301	51	0.5804	0.5783285672481079
150	301	301	52	0.562	0.5576657263351378
150	301	301	53	0.5774	0.5758434374191632
150	301	301	54	0.5828	0.5818822565401064
150	301	301	55	0.5679	0.5741636523100568
150	301	301	56	0.5778	0.5792506863311178
150	301	301	57	0.5722	0.5666444141399334
150	301	301	58	0.5999	0.6028532900375981
150	301	301	59	0.5837	0.5851464838668375
200	401	401	60	0.5414	0.5431322748896739

asked_length	QC_length	TQC_length	used_seed	th_fidelity	state_fidelity
200	401	401	61	0.5317	0.5326902379362701
200	401	401	62	0.5647	0.5671915603424857
200	401	401	63	0.5544	0.547922688969658
200	401	401	64	0.543	0.5453706323225868
200	401	401	65	0.5536	0.5573682160304024
200	401	401	66	0.5488	0.5544066741935936
200	401	401	67	0.5444	0.5397073829990505
200	401	401	68	0.5231	0.5269801904887417
200	401	401	69	0.528	0.5375991650341947
250	501	501	70	0.5399	0.5238675834937228
250	501	501	71	0.5163	0.5183457651067128
250	501	501	72	0.5207	0.5229835836972893
250	501	501	73	0.5177	0.5185604943351337
250	501	501	74	0.5317	0.5363232478862205
250	501	501	75	0.5238	0.5260356604133025
250	501	501	76	0.5238	0.5274960397466724
250	501	501	77	0.5183	0.5238007485709629
250	501	501	78	0.5317	0.526912255367247
250	501	501	79	0.5204	0.5262395496081861
300	601	601	80	0.513	0.5140612847872711
300	601	601	81	0.5028	0.5068345384460351
300	601	601	82	0.5054	0.5114674440354897
300	601	601	83	0.5097	0.5105467343981657
300	601	601	84	0.5114	0.5137467523796663
300	601	601	85	0.5049	0.5086695714454083
300	601	601	86	0.5084	0.5095359414028308
300	601	601	87	0.5139	0.5118180663686079
300	601	601	88	0.5115	0.5107579246820435
300	601	601	89	0.5121	0.5072696713888472
350	701	701	90	0.5179	0.5052251309649654
350	701	701	91	0.5151	0.5059428913605653
350	701	701	92	0.5149	0.5084461905003222
350	701	701	93	0.4993	0.5053058466392901
350	701	701	94	0.5016	0.5076293431946162
350	701	701	95	0.5094	0.505930877906788
350	701	701	96	0.4978	0.5066014856343606
350	701	701	97	0.5048	0.5064007356882035
350	701	701	98	0.4985	0.506099872896128
350	701	701	99	0.5087	0.5094012999418392

C.2 experimental implementation: RB results

The table below gives some general information about the generated sequences:

seq_count	pulse_count	total_pulse_time	total_angle	total_phase	used_seed
1	3	20000	2.0	2.0	0
2	4	20000	4.0	-0.5	1
3	3	20000	4.0	1.5	2
4	3	20000	2.0	2.5	3
5	6	40000	6.0	3.5	4
6	7	40000	8.0	4.0	5
7	6	40000	6.0	0.5	6
8	6	40000	6.0	3.5	7
9	9	50000	10.0	2.5	8
10	9	50000	10.0	2.5	9
11	8	50000	8.0	2.0	10
12	8	50000	8.0	5.0	11

C.3 Generated sequences

Generated sequences with Class parameters:
pulses = {"*pi_pulse*": 10000, "*pi/2_pulse*": 5000}
sequence_lengths = [3, 5, 7]
sample_amount = 4
seeds = list(range(12))

```
{
  'seq_1': {
    'rot_axis': ['Y', 'Y', 'X'],
    'pulse_time': [10000, 5000, 5000],
    'phase_shift': [0.5, 0.5, 1.0],
    'gates': ['P2', 'S3', 'S1']
  },
  'seq_2': {
    'rot_axis': ['-Y', 'Y', 'Y', '-X'],
    'pulse_time': [5000, 5000, 5000, 5000],
    'phase_shift': [-0.5, -0.5, -0.5, 1.0],
    'gates': ['I2', 'S3', 'S2']
  },
  'seq_3': {
    'rot_axis': ['Y', '-X', '-X'],
```

```
    'pulse_time': [10000, 5000, 5000],
    'phase_shift': [0.5, 0.0, 1.0],
    'gates': ['P2', 'S2', 'S2']
  },
  'seq_4': {
    'rot_axis': ['Y', 'X', 'X'],
    'pulse_time': [10000, 5000, 5000],
    'phase_shift': [0.5, 1.0, 1.0],
    'gates': ['P2', 'S1', 'S1']
  },
  'seq_5': {
    'rot_axis': ['X', '-Y', 'Y', 'Y', '-X', 'X'],
    'pulse_time': [10000, 5000, 10000, 5000, 5000, 5000],
    'phase_shift': [0.0, 0.5, 0.5, 0.5, 1.0, 1.0],
    'gates': ['P1', 'S4', 'P2', 'S3', 'I1']
  },
  'seq_6': {
    'rot_axis': ['X', '-Y', '-X', 'X', '-Y', '-X', 'X'],
    'pulse_time': [10000, 5000, 5000, 5000, 5000, 5000, 5000],
    'phase_shift': [0.0, 0.5, 1.0, 1.0, -0.5, 1.0, 1.0],
    'gates': ['P1', 'S4', 'I1', 'S4', 'I1']
  },
  'seq_7': {
    'rot_axis': ['-Y', 'Y', 'Y', 'X', '-X', 'X'],
    'pulse_time': [5000, 5000, 5000, 10000, 5000, 10000],
    'phase_shift': [-0.5, -0.5, -0.5, 0.0, 1.0, 1.0],
    'gates': ['I2', 'S3', 'P1', 'S2', 'P1']
  },
  'seq_8': {
    'rot_axis': ['Y', 'Y', 'X', '-Y', '-X', 'X'],
    'pulse_time': [10000, 5000, 10000, 5000, 5000, 5000],
    'phase_shift': [0.5, 0.5, 1.0, -0.5, 1.0, 1.0],
    'gates': ['P2', 'S3', 'P1', 'S4', 'I1']
  },
  'seq_9': {
    'rot_axis': ['X', '-X', '-X', 'X', '-Y', '-X', 'X', '-X', 'X'],
    'pulse_time': [10000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000],
    'phase_shift': [0.0, 1.0, 0.0, 0.0, -0.5, 1.0, 1.0, 0.0, 0.0],
    'gates': ['P1', 'S2', 'I1', 'S4', 'I1', 'S2', 'S1']
  },
  'seq_10': {
    'rot_axis': ['-Y', 'Y', '-Y', 'Y', '-X', '-X', 'X', 'Y', '-X'],
    'pulse_time': [5000, 5000, 5000, 10000, 5000, 5000, 5000, 5000, 5000],
```

```
    'phase_shift': [-0.5, -0.5, 0.5, 0.5, 1.0, 0.0, 0.0, 0.5, 1.0],
    'gates': ['I2', 'S4', 'P2', 'S2', 'I1', 'S3', 'S2']
  },
  'seq_11': {
    'rot_axis': ['Y', '-Y', '-Y', 'Y', 'X', 'Y', '-Y', 'X'],
    'pulse_time': [10000, 5000, 5000, 5000, 5000, 10000, 5000, 5000],
    'phase_shift': [0.5, -0.5, 0.5, 0.5, 0.0, -0.5, 0.5, 1.0],
    'gates': ['P2', 'S4', 'I2', 'S1', 'P2', 'S4', 'S1']
  },
  'seq_12': {
    'rot_axis': ['-X', 'X', 'X', 'Y', '-X', 'X', 'Y', '-X'],
    'pulse_time': [5000, 5000, 5000, 10000, 5000, 10000, 5000, 5000],
    'phase_shift': [1.0, 1.0, 1.0, 0.5, 0.0, 0.0, 0.5, 1.0],
    'gates': ['I1', 'S1', 'P2', 'S2', 'P1', 'S3', 'S2']
  }
}
```

Appendix D

Python Code

D.1 Simulations: Noise model

```
from ONEQRBClass import ONEQ_RBClass
from qiskit_aer import AerSimulator
from qiskit_aer.noise import NoiseModel, thermal_relaxation_error, depolarizing_error

# Operational time for the u3 native gate
time_u3 = 100
# Set up thermal relaxation error
u3_error = thermal_relaxation_error(99e4, 10e3, time_u3)

# Depolarizing error
dp_error = depolarizing_error(0.05, 1)

# Set up Noise model object
noise_thermal = NoiseModel()
noise_thermal.add_all_qubit_quantum_error(dp_error, ["SX", "u1"])
noise_thermal.add_quantum_error(u3_error, ["u3"], [0])

# Add Noise model to backend
backend = AerSimulator(method='density_matrix', noise_model=noise_thermal)
backend.set_options(noise_model=noise_thermal)
```

D.2 Simulations: ONEQ_RBClass

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display
from qiskit.circuit import QuantumCircuit
from qiskit import transpile
import qiskit.quantum_info as qi
from qiskit.providers.fake_provider import Fake1Q
from qiskit.circuit.library.standard_gates import (XGate, YGate, ZGate, HGate,
                                SGate, SdgGate, SXGate, SXdgGate)

pd.set_option('display.max_columns', None)
pd.set_option("display.max_rows", None)


class ONEQ_RBClass:

    def __init__(self, circuit_lengths=None, sample_amount=10, shots=int(1e4), seeds=None,
                 initial_state=None, backend=None, interleaved=False,
                 interleaved_gate=None, state_fidelity=False):
        """
        Initialize the class parameters for a
        one qubit class parameters Standard or Interleaved Randomized Benchmarking experiment.
        :param circuit_lengths: number of gates for each circuit (list of integers)
        (e.g.:[5, 10, 50, 100, 250])
        :param sample_amount: amount of executed circuits
        :param shots: amount of times a circuit will be executed (default: 1e4)
        :param seeds: list of seeds for each sequence
        :param initial_state: sets an initial state (dictionary)
        e.g.: {'0': QuantumCircuit(1)}
        :param backend: used backend
        :param interleaved: Set True for an interleaved experiment
        :param interleaved_gate: gate to be interleaved
        :param state_fidelity: Set True to return the process fidelity
        """

        if circuit_lengths is None:
            circuit_lengths = []

        self._circuit_lengths = circuit_lengths
        self._sample_amount = sample_amount
        self._shots = shots
        self._seeds = seeds

        if initial_state is None:
            initial_state = {'0': QuantumCircuit(1)}

        self._initial_state = initial_state

        if backend is None:
            backend = Fake1Q()
```

```
52
53          self._backend = backend
54
55          if interleaved is not None and interleaved_gate is None:
56              interleaved_gate = XGate
57
58          self._interleaved = interleaved
59          self._interleaved_gate = interleaved_gate
60          self._state_fidelity = state_fidelity
61          # Define dictionary to store circuit parameters and results
62          self._results = {"asked_length": [], "QC_length": [], "TQC_length": [], "used_seed": [],
63                           "meas_fidelity": [], "state_fidelity": []}
64
65      @property
66      def circuit_lengths(self):
67          """ Returns the circuit lengths """
68          return self._circuit_lengths
69
70      @circuit_lengths.setter
71      def circuit_lengths(self, circuit_lengths):
72          """
73          sets the given circuit lengths
74          """
75          if isinstance(circuit_lengths, list) and all(isinstance(item, int) for item in circuit_lengths):
76              self._circuit_lengths = circuit_lengths
77          else:
78              raise TypeError("'circuit_lengths' needs to be a list of integers.")
79
80      @property
81      def sample_amount(self):
82          """ Returns the sample amount """
83          return self._sample_amount
84
85      @sample_amount.setter
86      def sample_amount(self, sample_amount):
87          """
88          sets the given sample amount
89          """
90          if isinstance(sample_amount, int):
91              self._sample_amount = sample_amount
92          else:
93              raise TypeError("'sample_amount' needs to be an integer.")
94
95      @property
96      def shots(self):
97          """ Returns the shots """
98          return self._shots
99
100     @shots.setter
101     def shots(self, shots):
102         """
103         sets the given shots
104         """
```

```
105             if isinstance(shots, int):
106                 self._shots = shots
107             else:
108                 raise TypeError("'shots' needs to be an integer.")
109 
110         @property
111         def seeds(self):
112             """ Returns the seed """
113             return self._seeds
114 
115         @seeds.setter
116         def seeds(self, seeds):
117             """
118             sets the given seed
119             """
120             if seeds is None:
121                 self._seeds = seeds
122             elif isinstance(seeds, list) and all(isinstance(item, int) for item in seeds) and \
123                     len(seeds) == self.circuit_lengths * self.sample_amount:
124                 self._seeds = seeds
125             else:
126                 raise TypeError("'seeds' must be a list of integer "
127                                 "(length = circuit_lengths x sample_amount) or None.")
128 
129         @property
130         def initial_state(self):
131             """ Returns the initial state """
132             return self._initial_state
133 
134         @initial_state.setter
135         def initial_state(self, initial_state):
136             """
137             sets the given initial state
138             """
139             # Define initial string and circuit
140             init_string = list(initial_state.keys())[0]
141             ansatz = initial_state[init_string]
142 
143             if isinstance(initial_state, dict) and isinstance(init_string, str) and \\
144                 isinstance(ansatz, QuantumCircuit):
145                 self._initial_state = initial_state
146             else:
147                 raise TypeError('Initial state needs to be a circuit.QuantumCircuit
148                                                     and equal amount of qubits')
149 
150         @property
151         def backend(self):
152             """ Returns the given backend"""
153             return self._backend
154 
155         @property
156         def interleaved(self):
157             """ Returns the given boolean for interleaved experiment"""
```

```
158         return self._interleaved
159
160     @interleaved.setter
161     def interleaved(self, interleaved):
162         """
163         sets the given boolean for interleaved experiment:
164         """
165         if isinstance(interleaved, bool):
166             self._interleaved = interleaved
167         else:
168             raise TypeError('interleaved needs to be a boolean')
169
170     @property
171     def interleaved_gate(self):
172         """ Returns the given gate to be interleaved"""
173         return self._interleaved_gate
174
175     @interleaved_gate.setter
176     def interleaved_gate(self, interleaved_gate):
177         """
178         sets the given gate to be interleaved:
179         """
180         self._interleaved_gate = interleaved_gate
181
182     @property
183     def state_fidelity(self):
184         """ Returns the given boolean for state_fidelity measurements"""
185         return self._state_fidelity
186
187     @property
188     def results(self):
189         """ Returns the results """
190         return self._results
191
192     @staticmethod
193     def random_gate(rng):
194         """
195         Returns random gate
196         :param rng:random.default_rng object
197         :return: qiskit.circuit.library.standard_gates object
198         """
199         # List with one qubit gates
200         one_qubit_gates = [XGate, YGate, ZGate, HGate, SGate, SdgGate, SXGate, SXdgGate]
201         # Random gate choice
202         gate = rng.choice(one_qubit_gates)
203
204         return gate
205
206     def gate_sequences(self, length, seed):
207         """
208         standard gate sampling for a circuit with given length and seed.
209         :param seed: sequence seed
210         :param length: sequence length
```

```
211         :return: quantum circuit (qc)
212         """
213         # Initialize one qubit quantum circuit
214         qc = QuantumCircuit(1, 1)
215         # Define random.default_rng object
216         rng = np.random.default_rng(seed)
217 
218         if self.interleaved:
219             for i in range(length // 2):
220                 gate = self.random_gate(rng)
221                 qc.append(gate(None), [0])
222                 qc.append(self.interleaved_gate(None), [0])
223 
224             # Append another gate/interleaved gate if length is odd
225             if (length % 2) != 0:
226                 gate = self.random_gate(rng)
227                 qc.append(gate(None), [0])
228                 qc.append(self.interleaved_gate(None), [0])
229 
230         else:
231             for i in range(length):
232                 gate = self.random_gate(rng)
233                 qc.append(gate(None), [0])
234 
235         return qc
236 
237     def random_clifford_circuit(self, length, seed):
238         """
239         Returns transpiled and circuit with reversed circuit attached.
240         :param length: sequence length
241         :param seed: sequence seed
242         :return: transpiled quantum circuit (tqc)
243         """
244         # Random clifford circuit with given length and seed (interleaved if asked)
245         qc = self.gate_sequences(length, seed)
246 
247         # Get the inverse circuit
248         inv_ansatz = qc.inverse()
249         # Append to circuit
250         qc = qc.compose(inv_ansatz)
251 
252         # reverse to compose density matrix and initial state
253         qc = qc.reverse_ops()
254         if self.state_fidelity:
255             qc.save_density_matrix(label="rho1", conditional=True)
256 
257         # Retrieve initial circuit from dictionary and compose to circuit
258         init_string = list(self.initial_state.keys())[0]
259         init_qc = self.initial_state[init_string]
260         qc = qc.compose(init_qc)
261         # reverse again
262         qc = qc.reverse_ops()
263         # transpile circuit
```

```
264         tqc = transpile(qc, self.backend, optimization_level=0)
265
266         # Update results with circuit parameters
267         self.results["asked_length"].append(length)
268         self.results["QC_length"].append(len(qc))
269         self.results["TQC_length"].append(len(tqc))
270
271         return tqc
272
273     def measure_circuit(self, qc):
274         """
275         Function to measure the probability to obtain a certain state (initial state).
276         Calculates the 0th order fidelity of a given quantum circuit.
277         :param qc: The given quantum circuit.
278         :return: 0th order fidelity
279         """
280         qc.measure(0, 0)
281         # execute circuit and get results
282         job = self.backend.run(qc, shots=self.shots)
283         results = job.result()
284         # Get the initial state counts
285         # print(results.get_counts())
286         initial_state_counts = results.get_counts()['0']
287         # Calculate fidelity
288         fidelity = initial_state_counts / self.shots
289         print('measurement fidelity: ', fidelity)
290         # remove final measurements for density matrix state
291         qc.remove_final_measurements(inplace=True)
292         # update results
293         self.results["meas_fidelity"].append(fidelity)
294         return
295
296     def circuit_state_fidelity(self, qc):
297         """
298         Function calculates the circuit state fidelity
299         using two density matrices, the first after the initial state and
300         the second after the last gate.
301         :param qc: The given quantum circuit.
302         :return: circuit state fidelity
303         """
304
305         # set second density matrix
306         qc.save_density_matrix(label="rho2", conditional=True)
307         # execute circuit and get results
308         job = self.backend.run(qc)
309         results = job.result().data()
310         # Calculate state fidelity using the density matrices
311         state_fidelity = qi.state_fidelity(results['rho1'][''], results['rho2'][''])
312         print("state fidelity: ", state_fidelity)
313         # Update results
314         self.results["state_fidelity"].append(state_fidelity)
315         return
316
```

```
317     def sequence_sampler(self, length, count):
318         """
319         Generates and measures according to the asked samples
320         :param length: length af each circuit
321         :param count: ......
322         :return: Update results
323         """
324 
325         for sample in range(self.sample_amount):
326 
327             # Update sequence count
328             count += 1
329 
330             # Define a seed if given
331             if self.seeds is not None:
332                 seed = self.seeds[count - 1]
333             else:
334                 # Else define random seed for each circuit
335                 seed = np.random.randint(0, np.iinfo(np.int32).max)
336 
337             # update used sequence seed
338             self.results["used_seed"].append(seed)
339 
340             # Create circuit
341             qc = self.random_clifford_circuit(length, seed)
342             # Measure circuit
343             self.measure_circuit(qc)
344             if self.state_fidelity:
345                 self.circuit_state_fidelity(qc)
346 
347         return count
348 
349     def execute_randomized_benchmarking(self):
350         """
351         Execute randomized benchmarking experiment
352         :return: Update results
353         """
354         # Set count 0
355         count = 0
356 
357         for length in self.circuit_lengths:
358             # Sample sequences with given length
359             count = self.sequence_sampler(length, count)
360 
361         # Set up empty pandas dataframe for storing circuit parameters and fidelity results
362         results = pd.DataFrame(dict([(key, pd.Series(value)) for key, value in self.results.items()]))
363         display(results)
364 
365         # Save the dataframe to a CSV file
366         results.to_csv('IRB_AER_dp1_S_run1.csv', index=False)
367 
368         # Some plots
369         results.plot(kind='scatter', x='TQC_length', y='meas_fidelity', color='blue')
```

```
370         plt.xlabel("Number of native gates")
371         plt.ylabel("Measurement fidelity")
372         plt.title('1-qubit StandardRB experiment')
373         plt.grid()
374         plt.show()
375         return
```

D.3 Class ONEQ_SRB_experiment

```
import math
import numpy as np
import pandas as pd
from IPython.display import display
pd.set_option('display.max_columns', None)
pd.set_option("display.max_rows", None)


class ONEQ_SRB_experiment:

    def __init__(self, sequence_lengths=None, seeds=None, pulses=None, sample_amount=10,
                 shots=int(1e2), clock_time=1.):
        """
        Initialize the class parameters for a
        one qubit class parameters Standard Randomized Benchmarking experiment.
        :param sequence_lengths: amount of gates for each circuit (list of integers)
        (e.g.:[5, 10, 50, 100, 250])
        :param sample_amount: amount of executed circuits (optional: with given seeds for each sample
        :param shots: amount of times a circuit will be executed (default: 1e4)
        :param seeds: list of seeds for each sample
        (e.g.:[5, 10, 50, 100, 250])
        :param pulses: dictionary with all the pulses parameters
        (e.g.) {"pi_pulse": 100, "pi/2_pulse": 50}
        :param clock_time: time sampling resolution (default 1.0 ns)
        """
        if sequence_lengths is None:
            sequence_lengths = []
        if pulses is None:
            pulses = {"pi_pulse": None, "pi/2_pulse": None}

        self._sequence_lengths = sequence_lengths
        self._sample_amount = sample_amount
        self._shots = shots
        self._seeds = seeds
        self._clock_time = clock_time
        self._pulses = pulses
        self._sequences = {}
        self._current_phase_angle = 0.0
        self._results = {"seq_count": [], "pulse_count": [], "total_pulse_time": [], \\
                         "total_angle": [], "total_phase": [], "used_seed": []}

    @property
    def sequence_lengths(self):
        """ Returns the sequence lengths """
        return self._sequence_lengths

    @sequence_lengths.setter
    def sequence_lengths(self, sequence_lengths):
        """
        sets the sequence lengths
        """
```

```
52          # list of integers
53          if isinstance(sequence_lengths, list) and all(isinstance(item, int) \\
54              for item in sequence_lengths):
55              self._sequence_lengths = sequence_lengths
56          else:
57              raise TypeError("'sequence_lengths' needs to be a list of integers.")
58
59      @property
60      def sample_amount(self):
61          """ Returns the sample amount """
62          return self._sample_amount
63
64      @sample_amount.setter
65      def sample_amount(self, sample_amount):
66          """
67          sets the given sample amount
68          """
69          # integer
70          if isinstance(sample_amount, int):
71              self._sample_amount = sample_amount
72          else:
73              raise TypeError("'sample_amount' needs to be an integer.")
74
75      @property
76      def shots(self):
77          """ Returns the shots """
78          return self._shots
79
80      @shots.setter
81      def shots(self, shots):
82          """
83          sets the given shots
84          """
85          # integer
86          if isinstance(shots, int):
87              self._shots = shots
88          else:
89              raise TypeError("'shots' needs to be an integer.")
90
91      @property
92      def pulses(self):
93          """ Returns the pulse parameters """
94          return self._pulses
95
96      @pulses.setter
97      def pulses(self, pulses):
98          """
99          sets the x gates parameters
100         """
101         # dictionary without empty keys
102         if isinstance(pulses, dict) and None not in pulses.values():
103             self._pulses = pulses
104         else:
```

```
105             raise TypeError("'pulses' needs to be a dictionary without None values.")
106
107     @property
108     def seeds(self):
109         """ Returns the seed """
110         return self._seeds
111
112     @seeds.setter
113     def seeds(self, seeds):
114         """
115         sets the given seed
116         """
117         # list of integers or None
118         if seeds is None:
119             self._seeds = seeds
120         elif isinstance(seeds, list) and all(isinstance(item, int) for item in seeds):
121             self._seeds = seeds
122         else:
123             raise TypeError("'seeds' must be a list of integer or None.")
124
125     @property
126     def clock_time(self):
127         """ Returns the sequence lengths """
128         return self._clock_time
129
130     @property
131     def current_phase_angle(self):
132         """ Returns the current phase angle """
133         return self._current_phase_angle
134
135     @current_phase_angle.setter
136     def current_phase_angle(self, value=0):
137         self._current_phase_angle = value
138
139     @property
140     def sequences(self):
141         """ Return the sequences """
142         return self._sequences
143
144     @property
145     def results(self):
146         """ Return the results """
147         return self._results
148
149     @staticmethod
150     def sine_wave(times, frequency, amp, phase_shift):
151         return amp*np.sin(2 * np.pi * frequency * times + phase_shift)
152
153     @staticmethod
154     def cosine_wave(timec, frequency, amp, phase_shift):
155         return amp*np.cos(2 * np.pi * frequency * timec + phase_shift)
156
157     @staticmethod
```

```
158     def current_polar_angle(angle, phase=False):
159         n = math.floor(angle / 2)
160         final_angle = angle - 2 * n
161 
162         if phase and final_angle > 1.:
163             return -0.5
164         else:
165             return final_angle
166 
167     @staticmethod
168     def numeric_axis(axis):
169         """
170         Sets X axis to 0 and Y axis to 1
171         :param axis: X, Y, -X, -Y
172         :return: integer
173         """
174         if axis == 'X':
175             return 0
176         elif axis == 'Y':
177             return 1
178         elif axis == '-X':
179             return 2
180         elif axis == '-Y':
181             return 3
182         else:
183             raise NameError("Axis ", axis, " not recognised")
184 
185     def phase_shift(self, current_axis, new_axis, pulse_angle):
186         """
187         'phase_shift' calculates the phase shift needed between two gates
188         (for X and Y gates)
189         :param current_axis: X, Y, -X or -Y
190         :param new_axis: X, Y, -X or -Y
191         :param pulse_angle: current polar angle
192         :return: phase shift
193         """
194         if new_axis == "X" and new_axis == current_axis:
195             return self.current_phase_angle
196         if new_axis == "Y" and new_axis == current_axis:
197             return self.current_phase_angle
198 
199         current_axis = self.numeric_axis(current_axis)
200         new_axis = self.numeric_axis(new_axis)
201         current_angle = float(self.current_polar_angle(pulse_angle))
202 
203         if current_angle < 1.:
204             phases = [[0., 0.5, 1., -0.5], [0., 0.5, 1., -0.5]]
205         else:
206             phases = [[1., -0.5, 0., 0.5], [1., -0.5, 0., 0.5]]
207 
208         if self.current_phase_angle == phases[current_axis][new_axis]:
209             if new_axis == 2 or 3:
210                 self.current_phase_angle = self.current_polar_angle(phases[current_axis][new_axis] + 1.,
```

```
                                                                    phase=True)
        else:
            self.current_phase_angle = phases[current_axis][new_axis]

        return self.current_phase_angle

    def reverse_pulse(self, angle):
        """
        'reverse_pulse' gives the reverse pulse for a given final angle.
        Such that the qubit is back in the initial state.
        :param angle: final angle
        :return: pulse
        """
        # Inverse pulse
        final_angle = self.current_polar_angle(angle)

        if final_angle == 0:
            return "I1"
        elif final_angle == 0.5:
            return "S2"
        elif final_angle == 1:
            return "P1"
        elif final_angle == 1.5:
            return "S1"
        else:
            raise ValueError("The final gate ", final_angle, " is not a multiple of pi/2.")

    def transpiler(self, gate):
        """
        transpiler: Transpiles asked gate in the pulse sequences.
        :param gate: Asked gate to transpile in pulses.
        :return: pulse parameters (list of dictionaries)
        gate = {"type": 0, "pulse_time": p1_time, "phase_shift": 0}
        "type": 0 stands for X-gate, 1 stands for Y-gate
        "pulse_time": time interval for a given pulse
        "phase_shift": phase shift needed for given pulse, in units of pi (e.g. 1 = pi, 1/2 = pi/2)
        """
        ###################################
        # Pauli pulses (Pauli operations):
        ###################################
        p1_time = self.pulses["pi_pulse"]
        p1_gate = {"rot_axis": 'X', "pulse_angle": 1., "pulse_time": p1_time}

        p2_time = self.pulses["pi_pulse"]
        p2_gate = {"rot_axis": 'Y', "pulse_angle": 1., "pulse_time": p2_time}

        ########################################
        # pi/2 pulses (computational operations):
        ########################################
        s1_time = self.pulses["pi/2_pulse"]
        s1_gate = {"rot_axis": 'X', "pulse_angle": 0.5, "pulse_time": s1_time}

        s2_time = self.pulses["pi/2_pulse"]
```

```
264         s2_gate = {"rot_axis": '-X', "pulse_angle": 1.5, "pulse_time": s2_time}
265
266         s3_time = self.pulses["pi/2_pulse"]
267         s3_gate = {"rot_axis": 'Y', "pulse_angle": 0.5, "pulse_time": s3_time}
268
269         s4_time = self.pulses["pi/2_pulse"]
270         s4_gate = {"rot_axis": '-Y', "pulse_angle": 1.5, "pulse_time": s4_time}
271
272         if gate == "P1":
273             return [p1_gate]
274         elif gate == "P2":
275             return [p2_gate]
276         elif gate == "P5":
277             return [p2_gate, p1_gate]
278
279         elif gate == "S1":
280             return [s1_gate]
281         elif gate == "S2":
282             return [s2_gate]
283         elif gate == "S3":
284             return [s3_gate]
285         elif gate == "S4":
286             return [s4_gate]
287
288         ##################
289         # Identity gates:
290         ##################
291         elif gate == "I1":
292             return [s2_gate, s1_gate]
293         elif gate == "I2":
294             return [s4_gate, s3_gate]
295
296         ############################
297         # pi/2 pulse combinations:
298         ############################
299         # -X, Y, X = R2, R3, R1
300         elif gate == "R5":
301             return [s2_gate, s3_gate, s1_gate]
302         # -X, -Y, X = R2, R4, R1
303         elif gate == "R6":
304             return [s2_gate, s4_gate, s1_gate]
305
306         ##################
307         # Hadamard gates:
308         ##################
309         # X^2, Y = P3, R3
310         elif gate == "H1":
311             return [p1_gate, s3_gate]
312         # X^2, -Y = P3, R4
313         elif gate == "H2":
314             return [p1_gate, s4_gate]
315         # Y^2, X = P4, R1
316         elif gate == "H3":
```

```
317             return [p2_gate, s1_gate]
318         # Y^2, -X = P4, R2
319         elif gate == "H4":
320             return [p2_gate, s2_gate]
321         # X, Y, X = R1, R3, R1
322         elif gate == "H5":
323             return [s1_gate, s3_gate, s1_gate]
324         # -X, Y, -X = R2, R3, R2
325         elif gate == "H6":
326             return [s2_gate, s3_gate, s2_gate]
327 
328         else:
329             raise NameError("the asked gate", gate, " is not known")
330 
331     def pulse_sequence(self, length, seed):
332         """
333         Creates random pulse sequences
334         :return: pulse sequence
335         """
336         # Define pulse lists
337         pauli_list = ["I1", "I2", "P1", "P2"]
338         computational_list = ["S1", "S2", "S3", "S4"]
339 
340         # Create a dictionary to gather the sequence information
341         sequence_info = {"rot_axis": [], "pulse_time": [], "phase_shift": [], "gates": []}
342 
343         # Define random.default_rng object
344         rng = np.random.default_rng(seed)
345         # Set initial rot axis to X
346         current_rot_axis = 'X'
347         # Set total pulse time to zero
348         total_pulse_time = 0
349         # set pulse count to zero
350         pulse_count = 0
351         # set total angle to zero
352         total_angle = 0
353         # set total phase to zero
354         total_phase = 0
355         # Set current phase angle
356         self.current_phase_angle = 0.0
357 
358         for gate in range(length):
359             # Last gate is the reversed gate
360             if gate == length-1:
361                 pulse_string = self.reverse_pulse(total_angle)
362             elif gate % 2 == 0:
363                 # Choose random (according to seed) gate
364                 # Even gates -> paulis, odd gates -> computational
365                 pulse_string = rng.choice(pauli_list)
366             else:
367                 pulse_string = rng.choice(computational_list)
368 
369             # Append gate to sequence info
```

```
370             sequence_info["gates"].append(pulse_string)
371             # Transpile chosen gate to pulses
372             pulses = self.transpiler(pulse_string)
373 
374             for pulse in pulses:
375                 # Append the pulse count
376                 pulse_count += 1
377                 # Extract the pulse type
378                 sequence_info["rot_axis"].append(pulse["rot_axis"])
379                 # Extract the pulse time, update total pulse time
380                 sequence_info["pulse_time"].append(pulse["pulse_time"])
381                 # Extract the phase shift, update current pulse type
382                 phase_shift = self.phase_shift(current_rot_axis, pulse["rot_axis"], total_angle)
383                 total_pulse_time += pulse["pulse_time"]
384                 total_phase += phase_shift
385                 total_angle += pulse["pulse_angle"]
386                 # print(total_angle)
387                 sequence_info["phase_shift"].append(phase_shift)
388                 # Update current rotation axis
389                 if pulse["rot_axis"] == "-X":
390                     current_rot_axis = "X"
391                 elif pulse["rot_axis"] == "-Y":
392                     current_rot_axis = "Y"
393                 else:
394                     current_rot_axis = pulse["rot_axis"]
395 
396         # Update results
397         self.results["pulse_count"].append(pulse_count)
398         self.results["total_pulse_time"].append(total_pulse_time)
399         self.results["total_angle"].append(total_angle)
400         self.results["total_phase"].append(total_phase)
401 
402         return sequence_info
403 
404     def sequence_sampler(self, length, count):
405         """
406         Generates and measures according to the asked samples
407         :param length: length af each pulse sequence
408         :param count: sequence count
409         :return: Update results
410         """
411 
412         for sample in range(self.sample_amount):
413             # Update sequence count
414             count += 1
415 
416             # Define a seed if given
417             if self.seeds is not None:
418                 seed = self.seeds[count-1]
419             else:
420                 # Else define random seed for each sequence
421                 seed = np.random.randint(0, np.iinfo(np.int32).max)
422 
```

```
423             # update circuit parameters
424             self.results["used_seed"].append(seed)
425             self.results["seq_count"].append(count)
426             # Create pulse
427             sequence = self.pulse_sequence(length, seed)
428             # Make key and store sequence
429             key = "seq_" + str(count)
430             self.sequences[key] = sequence
431 
432         return count
433 
434     def build_randomized_benchmarking(self, display_results=False):
435         """
436         Execute randomized benchmarking experiment
437         :return: Update results
438         """
439         # Set count 0
440         count = 0
441 
442         for length in self.sequence_lengths:
443             # Sample circuits with given length
444             count = self.sequence_sampler(length, count)
445 
446         if display_results:
447             # Set up empty pandas dataframe for storing circuit parameters and fidelity results
448             results = pd.DataFrame(dict([(key, pd.Series(value)) for key, value in self.results.items
449             results.to_csv("name_file.csv", sep=',', index=False, encoding='utf-8')
450             display(results)
451 
452         return
453 
454     def build_arrays(self, seq_num):
455         """
456         Build sine and cosine waveforms for the I/Q channel
457         :param seq_num: sequence number
458         :return: sine/cosine custom pulse element
459         """
460         # Extract asked sequence
461         sequence = self.sequences[seq_num]
462         print(sequence)
463 
464         # Extract pulse times and phase shifts
465         times = sequence['pulse_time']
466         phases = sequence['phase_shift']
467 
468         # Total time
469         total_time = sum(times)*self.clock_time
470         # Set time numpy array
471         time_list = np.arange(0, total_time, self.clock_time)
472         # Define empty array
473         phase_list = np.array([])
474 
475         # Set up phase numpy array
```

```
476         for time, phase in zip(times, phases):
477             phase_slice = np.ones(time)*phase
478             phase_list = np.concatenate([phase_list, phase_slice])
479 
480         phase_list = phase_list*np.pi
481 
482         return time_list, phase_list, total_time
483 
484     def build_sine_waveform(self, duration, sample_rate=0.0001, amp=1.,seq_num="seq_1",freq=0.01):
485         """
486         build_sine_waveform
487         :return: sine waveform for IQ mixing
488         """
489         time_list, phase_list, total_time = self.build_arrays("seq_2")
490         sample_data = self.sine_wave(time_list, freq, amp, phase_list)
491         nbr_samples =int(duration)#/sample_rate
492 
493         if len(sample_data) < nbr_samples:
494             #pad the data out
495             sample_data = np.concatenate([sample_data, np.zeros(nbr_samples-len(sample_data))])
496         else:
497             #crash
498             raise Exception("Error")
499         return sample_data
500 
501     def build_cosine_waveform(self, duration, sample_rate=0.0001, amp=1.,seq_num="seq_2",freq=0.01):
502         """
503         build_cosine_waveform
504         :return: cosine waveform for IQ mixing
505         """
506         time_list, phase_list, total_time = self.build_arrays("seq_2")
507         sample_data = self.cosine_wave(time_list, freq, amp, phase_list)
508         nbr_samples = int(duration)#/sample_rate
509 
510         if len(sample_data) < nbr_samples:
511             #pad the data out
512             print(np.zeros(nbr_samples-len(sample_data)))
513             sample_data = np.concatenate([sample_data, np.zeros(nbr_samples-len(sample_data))])
514         else:
515             #crash
516             raise Exception("Error")
517 
518         return sample_data
```

Bibliography

[1] Anton Robert, Panagiotis Kl. Barkoutsos, Stefan Woerner, and Ivano Tavernelli. Resource-efficient quantum algorithm for protein folding. *npj Quantum Information*, 7(1), February 2021.

[2] Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences*, 114(29):7555–7560, July 2017.

[3] Guido Burkard, Thaddeus D. Ladd, Andrew Pan, John M. Nichol, and Jason R. Petta. Semiconductor spin qubits. *Reviews of Modern Physics*, 95(2), June 2023.

[4] Erika Kawakami. Characterization of an electron spin qubit in a si/SiGe quantum dot. [Dissertation (TUDelft), Delft University of Technology]. ISBN: 9789085932666 http://resolver.tudelft.nl/uuid:dd0886f0-2f3a-421d-ac92-bdc4da5985b5.

[5] W. I. L. Lawrie. (2022). spin qubits in silicon and germanium. [Dissertation (TUDelft), Delft University of Technology]. https://doi.org/10.4233/uuid:97c4ea24-9672-4e0b-b7a5-e3a48258c871.

[6] S. G. J. Philips. (2023). scaling a spin qubit quantum processor from two to six qubits. [Dissertation (TUDelft), Delft University of Technology]. https://doi.org/10.4233/uuid:62dcc7a6-df92-43e1-b40c-9879f97eabb6.

[7] Xioa Xue. Performance benchmarking of silicon quantum processors. [Dissertation (TUDelft), Delft University of Technology]. ISBN: 9789085932666 https://doi.org/10.4233/uuid:20fce6ef-6bb3-42a1-bdd3-53b2a282f0ae.

[8] X. Xue, T.F. Watson, J. Helsen, D.R. Ward, D.E. Savage, M.G. Lagally, S.N. Coppersmith, M.A. Eriksson, S. Wehner, and L.M.K. Vandersypen. Benchmarking gate fidelities in a two-qubit device. *Physical Review X*, 9(2), April 2019.

[9] Adam R. Mills, Charles R. Guinn, Michael J. Gullans, Anthony J. Sigillito, Mayer M. Feldman, Erik Nielsen, and Jason R. Petta. Two-qubit silicon quan-

tum processor with operation fidelity exceeding 99 *Science Advances*, 8(14), April 2022.

[10] J. M. Chow, J. M. Gambetta, L. Tornberg, Jens Koch, Lev S. Bishop, A. A. Houck, B. R. Johnson, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf. Randomized benchmarking and process tomography for gate errors in a solid-state qubit. *Physical Review Letters*, 102(9), March 2009.

[11] Daniel Greenbaum. Introduction to quantum gate set tomography, 2015.

[12] E. Knill, D. Leibfried, R. Reichle, J. Britton, R. B. Blakestad, J. D. Jost, C. Langer, R. Ozeri, S. Seidelin, and D. J. Wineland. Randomized benchmarking of quantum gates. *Physical Review A*, 77(1), January 2008.

[13] Joel J. Wallman. Randomized benchmarking with gate-dependent noise. *Quantum*, 2:47, January 2018.

[14] Timothy Proctor, Kenneth Rudinger, Kevin Young, Mohan Sarovar, and Robin Blume-Kohout. What randomized benchmarking actually measures. *Physical Review Letters*, 119(13), September 2017.

[15] Toshinari Itoko and Rudy Raymond. Sampling strategy optimization for randomized benchmarking, 2021.

[16] Easwar Magesan, Jay M. Gambetta, B. R. Johnson, Colm A. Ryan, Jerry M. Chow, Seth T. Merkel, Marcus P. da Silva, George A. Keefe, Mary B. Rothwell, Thomas A. Ohki, Mark B. Ketchen, and M. Steffen. Efficient measurement of quantum gate error by interleaved randomized benchmarking. *Physical Review Letters*, 109(8), August 2012.

[17] Conrad Strydom and Mark Tame. Measurement-based interleaved randomised benchmarking using ibm processors. *Physica Scripta*, 98(2):025106, January 2023.

[18] IBM. Ibm quantum computing, 2023. consulted on 27/09/2023 from https://www.ibm.com/quantum.

[19] prof. Francois Peeters prof. Lucian Covaci. Lecture notes in low dimensional physics (university of antwerp), September 2023.

[20] Charley, s. from bits to qubits | physics. University of California, Berkeley (January 13, 2022) Retrieved from: https://physics.berkeley.edu/news-events/news/from-bits-to-qubits.

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

[22] T2 vs t2* what is the difference between t2 and t2*. Questions and Answers in MRI Retrieved from: https://mriquestions.com/t2-vs-t2.html.

[23] Zhao Fen Chuang Isaac. Lecture 19: How to build your own quantum computer, November 2003. Department of Mathematics, MIT, Retrieved from: https://ocw.mit.edu/courses/18-435j-quantum-computation-fall-2003/c95a265dc90e89f2323d84be985fd322_qc_lec19.pdf.

[24] Einstein Relatively Easy. C-NOT gate, bell state and entanglement. https://einsteinrelativelyeasy.com/index.php/quantum-mechanics/156-c-not-gate-bell-state-and-entanglement.

[25] Falk Peter Unger. (2008). noise in quantum and classical computation and non-locality. Dissertation Institute for logic, language, and computation (Universiteit Amsterdam) https://eprints.illc.uva.nl/id/eprint/2066/.

[26] Boone, Kristine. *Concepts and methods for benchmarking quantum computers*. PhD thesis, 2021. (UWSpace) Retrieved from: http://hdl.handle.net/10012/17139.

[27] Peter Balazs. Hilbert-schmidt operators and frames - classification, approximation by multipliers and algorithms, 2006.

[28] D Keith, S K Gorman, L Kranz, Y He, J G Keizer, M A Broome, and M Y Simmons. Benchmarking high fidelity single-shot readout of semiconductor qubits. *New Journal of Physics*, 21(6):063011, jun 2019.

[29] Magesan, Easwar. Gaining information about a quantum channel via twirling. Master's thesis, 2008. (UWSpace) Retrieved from: http://hdl.handle.net/10012/3828.

[30] Adam M. Meier. Randomized benchmarking of clifford operators, 2018.

[31] Christoph Dankert, Richard Cleve, Joseph Emerson, and Etera Livine. Exact and approximate unitary 2-designs and their application to fidelity estimation. *Physical Review A*, 80(1), July 2009.

[32] Magesan, Easwar. *Characterizing Noise in Quantum Systems*. PhD thesis, 2012. (UWSpace) Retrieved from: http://hdl.handle.net/10012/6832.

[33] Easwar Magesan, Jay M. Gambetta, and Joseph Emerson. Characterizing quantum gates via randomized benchmarking. *Physical Review A*, 85(4), April 2012.

[34] Easwar Magesan, J. M. Gambetta, and Joseph Emerson. Scalable and robust randomized benchmarking of quantum processes. *Physical Review Letters*, 106(18), May 2011.

[35] Samuel Haberthur. *Randomized Benchmarking of Two-Qubit Gates*. Master thesis, Department of physics - Laboratory for Solid State Physics - Quantum Device Lab, August 2015.

[36] IBM. Build noise models, 2023. (IBM Quantum Documentation). consulted on 28/11/2023 from https://docs.quantum.ibm.com/verify/building_noise_models.

[37] IBM. Get backend information with qiskit, 2023. (IBM Quantum Documentation). consulted on 03/12/2023 from https://docs.quantum.ibm.com/run/get-backend-information.

[38] M. Shane Burns. Error analysis for introductory physics labs, November 2013.

[39] Claus Fuhner. *Magneto-Transport Investigations on Multi-Electron Quantum Dots: Coulomb Blockade, Kondo Effect, and Fano Regime*. Phd thesis, University of Hannover, October 2002. Available at https://edocs.tib.eu/files/e01dh03/359800890.pdf.

[40] Andreas Fuhrer and Carina Fasth. Lecture notes in coulmb blockade in quantum dots, April 2007. Available at https://www.ftf.lth.se/fileadmin/ftf/Course_pages/FFF042/cb_lecture07.pdf.

[41] Nard Dumoulin Stuyck. *Design and Characterization of Quantum Silicon-Based Devices for Semiconducting Qubit Implementation*. Phd thesis, KU Leuven, September 2021.

[42] Arnout Beckers, Armin Tajalli, and Jean-Michel Sallese. A review on quantum computing: Qubits, cryogenic electronics and cryogenic mosfet physics, 08 2019.

[43] Ke Wang, Hai-Ou Li, Ming Xiao, Gang Cao, and Guo-Ping Guo. Spin manipulation in semiconductor quantum dots qubit*. *Chinese Physics B*, 27(9):090308, sep 2018.

[44] Lecture 16. C/cs/phys 191 spin manipulation ii (resonance), quantum gates on spins, October 2005. https://inst.eecs.berkeley.edu/~cs191/fa05/lectures/lecture16_fa05_1.pdf.

[45] Graeme Hanson Takeji Takui, Lawrence Berliner. *Electron Spin Resonance (ESR) Based Quantum Computing*. Springer, 2016.

[46] Gary Wolfowicz and John Morton. *Pulse Techniques for Quantum Information Processing*, volume 5, pages 1515–1528. 12 2016.

[47] Prof. Ted Jacobson. Spin resonance—phys623—, March 2017. University of Maryland. Retrieved from: https://www.physics.umd.edu/grt/taj/623e/623ehw3.pdf.

[48] Quantum Village. Rabi oscillations and single quantum gates. (2023)[Video].YouTube. Consulted on 20/04/2024 from https://www.youtube.com/watch?v=Gc0QfobJAdg.

[49] C A Ryan, M Laforest, and R Laflamme. Randomized benchmarking of single- and multi-qubit control in liquid-state nmr quantum information processing. *New Journal of Physics*, 11(1):013034, January 2009.

[50] Daniel K. Park, Guanru Feng, Robabeh Rahimi, Jonathan Baugh, and Raymond Laflamme. Randomized benchmarking of quantum gates implemented by electron spin resonance. *Journal of Magnetic Resonance*, 267:68–78, June 2016.

[51] prof. Wim Wenseleers. Lecture notes in experimentele technieken: signaalverwerking, vacuum en lage temperaturen (university of antwerp), September 2021.

[52] Antonio Rubio Abadal. Calibration of an iq mixer for continuous and pulsed modulation, August 2014. Available at https://qudev.phys.ethz.ch/static/content/science/Documents/semester/Antonio_RubioAbadal_SemesterThesis_140805.pdf.

[53] Simon Schmidlin. Generation of amplitude and phase controlled microwave pulses for qubit manipulation in circuit qed, March 2009. Available at https://qudev.phys.ethz.ch/static/content/science/Documents/master/Schmidlin_Simon_MasterThesis.pdf.

[54] Keysight Technologies. Comparing arbitrary waveform vs function generators: A deep dive - keysight technologies. (2024) Retrieved from: https://saving.em.keysight.com/en/used/knowledge/guides/arbitrary-waveform-generator-vs-function-generator.

[55] Jay M. Gambetta, A. D. Córcoles, S. T. Merkel, B. R. Johnson, John A. Smolin, Jerry M. Chow, Colm A. Ryan, Chad Rigetti, S. Poletto, Thomas A. Ohki, Mark B. Ketchen, and M. Steffen. Characterization of addressability by simultaneous randomized benchmarking. *Physical Review Letters*, 109(24), December 2012.