POLITECNICO DI TORINO

Master's Degree in Electronic Engineering

Master's Degree Thesis

Solid-State Simulations Advancing Conventional Processes Toward Spin Qubit Technologies

Supervisors
prof. Gianluca PICCININI
prof. Mariagrazia GRAZIANO
doc. Nicola CARBONETTA

Candidate
Lorenzo BARBERO

Academic Year 2024-2025

Summary

Quantum computers are emerging as a revolutionary technology capable of performing certain computations far more efficiently than classical computers. By exploiting the principles of quantum mechanics, such as superposition and entanglement, these devices can address problems that are currently beyond the reach of classical computers, including complex optimization, large-scale data analysis, and molecular simulations. Quantum computing has the potential to accelerate the development of new materials and pharmaceuticals, improve financial modelling, enhance cybersecurity and support advances in artificial intelligence. The investigation of state-of-the-art devices as potential platforms for spin qubits is motivated by the pursuit of novel technological solutions that can deliver quantum computing at lower cost, while retaining the remarkable performance characteristics of such devices.

The goal of this work is to provide a versatile methodology for studying the behaviour of custom semiconductor devices as hosts for spin qubits, achieved by engineering gate-defined quantum dots. A key aspect is the investigation of approaches grounded in widely adopted technological processes, thereby leveraging the existing capabilities of mature and well-established manufacturing facilities. To this end, the primary technological architecture considered is the planar Metal-Oxide-Semiconductor (MOS) on a Silicon-On-Insulator (SOI) wafer substrate. This highly mature platform can be realized through well-known processes that are extensively documented in the scientific literature and remain feasible even for low-volume research laboratories. Several devices were modelled and simulated in this work. While the primary focus was on planar devices fabricated on SOI wafer substrates, the study was also extended to three-dimensional structures such as fins and Gate-All-Around (GAA) nanowires. A three-dimensional representation of each device geometry was first developed to define and analyze a given architecture, and subsequently used for simulations of both static and dynamic confinement behaviour.

The primary tool employed for solid-state simulations is QTCAD, a quantum technology simulation platform developed by Nanoacademic Technologies Inc. It enables finite-element modelling based on suitable mesh files, in which each volume and relevant surfaces are labelled within the open-source CAD software Gmsh to assign physical properties such as materials and gate contacts. The geometries for each device are modelled in Fusion, a versatile CAD software from Autodesk, in this work used under a personal non-commercial licence. It enables the extrusion of solid geometries from two-dimensional sketches drawn parametrically, allowing the use of variables that automatically update the entire geometry when modified. The customizable nature of the exploited tools enabled the development of a complete workflow to simulate custom-defined devices and to post-process the obtained results, thereby allowing the study of parameter dependencies. Once quantum confinement was verified, the fabrication process was simulated with Synopsys Sentaurus

Process in order to more accurately reflect realistic manufacturing scenarios. The devices were designed with consideration for potential fabrication at the PiQuET research laboratory in Turin, in collaboration with Politecnico di Torino. To account for this aspect, the laboratory's capabilities were taken into account to better model the fabrication simulation. Seven devices were defined: a planar single quantum dot (SQD) and double quantum dot (DQD), along with two variants incorporating modelled doping profiles (obtained by SProcess doping simulations via thermal diffusion) within the source/drain contacts; a SQD FinFET-like structure; and both SQD and DQD Nanowire GAAFET-like structures. Each device provided clear evidence that the confinement and transport behaviour are satisfactory and consistent with expectations, demonstrating that the developed workflow performs as intended and constitutes a practical tool for investigating the suitability of these devices as platforms for hosting spin qubits.

In the first part of this work, the theoretical concepts that form the foundation of the study are reviewed. The discussion begins with the fundamentals of quantum mechanics, then moves to the principles of spin qubits in solid-state systems, and finally addresses how gate-defined quantum dots can be engineered and how state-of-the-art devices implement these concepts. The second part of this work is dedicated to the implementation and evaluation phase, where the methodology for simulating both the quantum behaviour and the fabrication processes is explained in detail. This section not only explains the simulation workflow step by step but also provides a structured reference to ensure the reproducibility of the results.

Acknowledgements

Firstly, I would like to thank prof. Gianluca Piccinini, who, thanks to the Integrated System Technology course at Politecnico di Torino, taught me the fundamental concepts about semiconductor devices and state-of-the-art fabrication processes for CMOS/VLSI and introduced and encouraged me into the complex field of quantum computing and spin qubits. My gratitude also extends to prof. Mariagrazia Graziano, who supervised this work and provided guidance in the design and study of nanodevices for quantum applications. Special thanks go to the doctoral researcher Nicola Carbonetta, who provided invaluable assistance throughout all stages of this work, supervising both the simulation of the fabrication processes and the writing of this thesis. I wish to recognize the contribution of Leonardo Ossino, who assisted with the fabrication process simulations as part of his internship at the Electronics and Telecommunications Department (DET) of Politecnico di Torino. I wish to thank the postdoctoral researcher Fabrizio Mo, who offered valuable help in the early stages of this work with technical support and research of relevant study material. In addition, I wish to thank Yuri Ardesi for guidance regarding the formalities and regulatory aspects of this thesis.

I would like to conclude by expressing my heartfelt thanks to my family and friends. Thanks to my beloved parents, Katia and Mauro, my sister, Sara, and my beloved grandmother, Anna, who always encouraged me to pursue my goals to the fullest and supported me throughout all my studies. My deepest thanks to Chiara, for her kindness, patience and support throughout our studies together, who has been a true companion to me throughout this journey.

Contents

List of Figures

List of Acronyms

AFM Atomic Force Microscope.

ALD Atomic Layer Deposition.

ALE Atomic Layer Etching.

API Application Programming Interface.

CMOS Complementary Metal-Oxide-Semiconductor.

CMP Chemical-Mechanical Polishing.

CNOT Controlled-NOT Gate.

CSV Comma-Separated Values.

CVD Chemical Vapor Deposition.

DPI Dots Per Inch.

DQD Double Quantum Dot.

EBL Electron Beam Lithography.

EDSR Electric Dipole Spin Resonance.

EPR Einstein-Podolsky-Rosen.

ESR Electron Spin Resonance.

FD Fully Depleted, or Fermi-Dirac.

FEM Finite Element Method.

FET Field-Effect Transistor.

FIB Focused Ion Beam.

FID Free Induction Decay.

FVM Finite Volume Method.

GAA Gate-All-Around.

ICP Inductively Coupled Plasma.

IEEE Institute of Electrical and Electronics Engineers.

IGES Initial Graphics Exchange Specification.

KMC Kinetic Monte Carlo.

LPCVD Low-Pressure Chemical Vapor Deposition.

MOS Metal-Oxide-Semiconductor.

NEGF Non-Equilibrium Green's Function.

NURBS Non-Uniform Rational B-Splines.

NV Nitrogen Vacancy.

PDF Probability Density Function.

PEO Plasma Enhanced Oxidation.

PVD Physical Vapor Deposition.

QD Quantum Dot.

QPC Quantum Point Contact.

QTCAD Quantum Technology Computer-Aided Design.

QW Quantum Well.

RIE Reactive Ion Etching.

RMG Replace Metal Gate.

SEM Scanning Electron Microscope.

SET Single Electron Transistor.

SI International System of Units.

SMTP Simple Mail Transfer Protocol.

SOI Silicon-On-Insulator.

SOTA	State Of The Art.
SQD	Single Quantum Dot.
SSH	Secure Shell.
STEP	Standard for the Exchange of Product model data.
STI	Shallow Trench Isolation.
TCAD	Technology Computer-Aided Design.
TEM	Transmission Electron Microscope.
TFET	Tunnel Field-Effect Transistor.
UV	Ultraviolet.
VLSI	Very-Large-Scale Integration.

Part I

Theoretical Background

Chapter 1

Introduction to Quantum Mechanics for Nanoscale Devices

In this first chapter, the fundamental principles of quantum mechanics will be revisited. Rather than offering a comprehensive treatment of the entire subject, this section aims to provide a conceptual foundation to support the understanding of phenomena that emerge in nanostructured electronic devices. This preliminary analysis is intended to prepare the ground for more advanced and application-oriented topics that will be discussed in the following chapters.

1.1 Principles of Quantum Mechanics

"Quantum mechanics was developed in the early decades of the 20th century, driven by the need to explain phenomena that, in some cases, had been observed in earlier times. Scientific inquiry into the wave nature of light began in the 17th and 18th centuries, when scientists such as Robert Hooke, Christiaan Huygens and Leonhard Euler proposed a wave theory of light based on experimental observations." [3] Quantum mechanics is the fundamental physical theory that describes the behaviour of physical systems at atomic and subatomic scale. The need to branch from classical mechanics laws rises from the fact that the latter model fails to provide accurate results at such scale. Unlike classical physics, quantum mechanics addresses the probabilistic and discrete nature of the physical quantities it describes. If we think of a physical object at a macroscopic scale, it is possible to compute precise results about it's behaviour based on continuos and deterministic variables, while, if it's needed to study the trajectory or the position of a submicroscopic particle, like an electron or a photon, probability and discretized variables comes into play. Although quantum mechanics is generally associated with the study of microscopic systems like atoms, molecules and subatomic particles, experimental evidence has shown that its principles can also apply to larger structures, including complex molecules made of thousands of atoms [18]. This extension suggests that classical mechanics can actually be seen as an approximation of quantum mechanics, valid only within everyday macroscopic conditions [24].

A fundamental feature of quantum mechanics is that usually it is not possible to exactly predict with absolute certainty the outcome you are looking to calculate, but only to get a probability of what will happen. This probability can be observed in a function called *probability amplitude*, mathematically found by taking the square root of the absolute value of a complex number. The German physicist Max Born formulated and published in 1926 one of the key postulates of quantum mechanics known as the Born rule, used to get the probability density of a physical system based on measurements on it. These measurements directly gives results of the quantum state the systems belongs, described in physics by a complex number called wavefunction and usually denoted by the Greek letter ψ (lower-case psi). According to the de Broglie hypothesis, every object in the universe is associated with a wave. Thus every object, from an elementary particle to atoms, molecules and on up to planets and beyond are subject to the uncertainty principle.
Born rule says that if we have a normalized system wavefunction, depending for example upon position coordinates *(x,y,z)* and a temporal coordinate t, the probability density p for the result of a measurement of the system's position at time t_0 is [8]

$$p\left(x, y, z, t_0\right) = \left|\psi\left(x, y, z, t_0\right)\right|^2 \tag{1.1}$$

In probability theory, this is called a probability density function (PDF) and can be integrated from two position points a and b in order to get the probability of finding, for example a subatomic particle, between the two points along a specific direction

$$\mathrm{P}[a \leq X \leq b] = \int_a^b |\psi(x)|^2 \, \mathrm{d}x \tag{1.2}$$

One of the consequences derived from the mathematical structure of quantum mechanics is a limitation in the simultaneous predictability of certain physical properties. This concept is expressed by Heisenberg's indeterminacy principle, often referred to as the uncertainty principle, originally formulated in 1927 by the German physicist Werner Heisenberg. The principle states that specific pairs of physical quantities, known as complementary or canonically conjugate variables, such as position and momentum, cannot both be measured with unlimited precision. The more accurately one of the two variables is known, the less precisely the other can be determined.
Formally, the indetermination rises from the formal inequality relating the standard deviation of position σ_x and the standard deviation of momentum σ_p [55]

$$\sigma_x \sigma_p \geq \frac{\hbar}{2} \tag{1.3}$$

where $\hbar = \frac{h}{2\pi}$ is the reduced Plank constant.

1.1.1 Quantum Interference and the Double Slit Experiment

Another result that comes from the mathematical framework of quantum mechanics is quantum interference, which reflects the wave-like behaviour of particles. When a particle or system can reach the same final state through multiple indistinguishable paths, the probability amplitudes of each path add together. The actual probability is given by the square of the total amplitude, so the paths can interfere with each other, either strengthening the result (constructive interference) or cancelling it out (destructive interference), depending on how their phases relate. This phenomenon only shows up at the submicroscopic scale and has no direct equivalent in classical physics, playing a central role in experiments like the double-slit experiment.

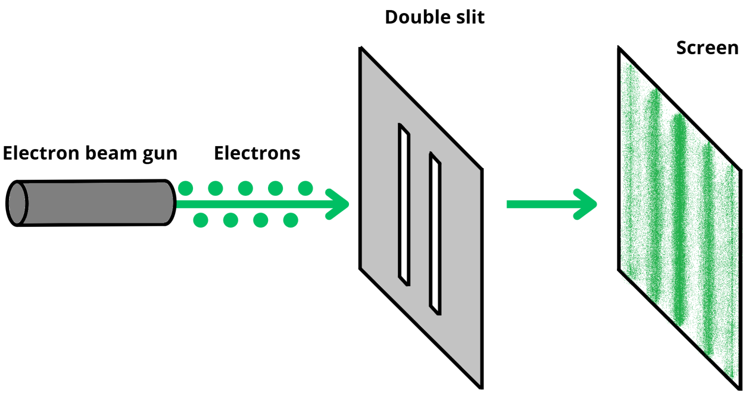


Figure 1.1: Double slit experiment

The experiment setup includes a source that emits particles, like electrons or photons, toward a barrier with two narrow slits and a detection screen behind the slits. When both slits are open and no measurement is made to find out which slit the particle goes through, an interference pattern appears on the screen, showing alternating areas of high and low detection probability, similar to what happens with classical waves. This suggests that each particle acts as if it passes through both slits at the same time, interfering with itself. However, when a measurement is made to determine the actual slit the particle goes through, the interference pattern disappears and the result looks like classical particles going through one slit or the other. This change depending on whether a measurement is performed shows that quantum particle behavior cannot be fully explained by classical mechanics.

The double-slit experiment thus played a crucial role in the discovery and understanding of wave-particle duality, the idea that quantum entities, such as electrons and photons, exhibit both wave-like and particle-like properties, depending on how they are observed.

1.1.2 Quantum Tunnelling

Another non-classical phenomenon predicted by quantum mechanics is quantum tunnelling, where a particle that goes up against a potential barrier has a non-zero probability to cross it, even if its kinetic energy is smaller than the maximum of the potential. This effect arises from the wave nature of particles in quantum mechanics.

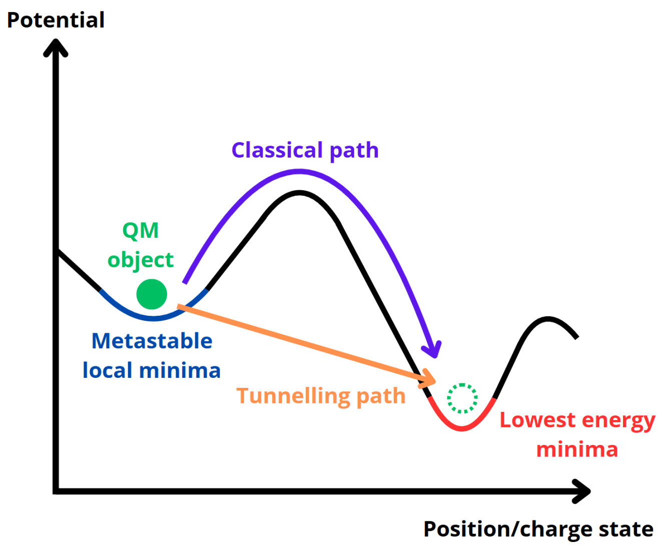


Figure 1.2: Simplified diagram of quantum tunnelling

In microelectronics, quantum tunnelling plays both positive and negative roles. It is used in devices like tunnel diodes, where it allows very fast current flow and quick switching, making them ideal for some high-frequency applications. In Tunnel Field-Effect Transistors (TFETs), tunnelling is exploited to enable low-power operation, as electrons tunnel through a potential barrier at the source-channel junction. This leads to a steep subthreshold slope and lower power use compared to traditional MOSFETs.

As devices continue to shrink, tunnelling also brings challenges. In modern transistors, especially those with gate lengths of just a few nanometers, unwanted tunnelling currents cause leakage, increasing power consumption and reducing efficiency. This issue limits the performance and scaling of standard CMOS devices. Addressing tunnelling-related problems is a major focus in developing future microelectronic technologies, through new materials or innovative transistor designs.

1.1.3 Quantum Entanglement

Quantum entanglement is among the most fascinating and fundamental phenomena in quantum mechanics. It was initially described by Albert Einstein, Boris Podolsky and Nathan Rosen in 1935 through the well-known EPR paradox [16], which challenged whether quantum mechanics could fully describe physical reality. Later experiments, especially those conducted by John Bell in the 1960s, offered strong evidence that quantum entanglement is indeed a real and non-local effect.

It is the phenomenon where the quantum state of each particle in a group cannot be described independently from the others, even when the particles are separated by large distances. Measurements of properties like position, momentum, spin and polarization on entangled particles can sometimes show perfect correlations. Although these correlations happen instantly, even between particles light years apart, they cannot be used to send information faster than light. This is explained by the no-communication theorem, which states that during the measurement of an entangled state, one observer cannot transmit information to another, no matter how far apart they are. The inherent randomness of quantum measurements prevents using entanglement for faster-than-light communication, preserving causality and aligning with special relativity.

Mathematically, quantum entanglement is often described using a superposition of states. One of the most common entangled states of two particles is the Bell state. In Bra-ket notation, one of the four Bell states (also known as an EPR state) [32] is:

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \tag{1.4}$$

This represents an entangled state in which, for example, two electrons are correlated: if one is measured in the state $|0\rangle$, the other will be in $|1\rangle$, and vice versa, each with 50% probability. The measurement outcomes are probabilistic, with the probability of each result determined by the squared magnitudes of the coefficients of each possible state (here $\frac{1}{\sqrt{2}}$ for both, so that the total probability is 1 without the need to normalize). The superposition state holds until the presence of an observer. Here the wavefunction collapses to one defined state, allowing the measurement to be made and so determine the quantum state at the instant the measurement is carried out.

Entanglement plays a key role in several quantum technologies, such as quantum computing, quantum cryptography and quantum teleportation. It enables faster computation, secure communication and the transfer of quantum states over distances. Despite being counter-intuitive, quantum entanglement has been confirmed through experiments and remains a central subject in both theoretical and experimental quantum research.

1.2 Schrödinger's Wave Equation

"The Schrödinger's equation is a partial differential equation that governs the wave function of a non-relativistic quantum-mechanical system."[19] It describes how the quantum state of a particle evolves over time, assuming the particle moves much slower than the speed of light. This equation allows us to calculate the probability of finding a particle in a certain position and understand its behaviour at microscopic scales, where classical mechanics no longer applies. The equation is named after Erwin Schrödinger, an Austrian physicist who formulated it in 1925 and published it in 1926. This work laid the foundation for research that earned him the Nobel Prize in Physics in 1933 [48][38]. Conceptually, the Schrödinger equation serves as the quantum equivalent of Newton's second law in classical mechanics.

The equation can take different forms depending on the context, but the most general one is the time-dependent Schrödinger equation, which describes how a system evolves over time [40]:

$$i\hbar\frac{\partial}{\partial t}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle \tag{1.5}$$

$|\Psi(t)\rangle$ is the time-dependent state vector of the quantum system, defined in a separable complex *Hilbert space* $\mathcal{H}$, a vector space equipped with an inner product operation, postulated to be normalized such that $\langle\psi,\psi\rangle = 1$. It is possible to relate this value to the system wavefunction, which is the representation of the state vector in a specific basis, for example, the position basis

$$\psi(x,t) = \langle x|\psi(t)\rangle \tag{1.6}$$

The wavefunction so is the projection of the state vector onto the position vector $|x\rangle$. If $|\psi(x)\rangle$ is known, the state vector can be retrieved by doing the inverse of this projection, by taking the superposition over all possible solutions. This translates to an integration over the position vector

$$|\Psi(t)\rangle = \int \psi(x,t)|x\rangle dx \tag{1.7}$$

$\hat{H}$ is the Hamiltonian of the system, an operator corresponding to the total energy of that system, including both kinetic energy and potential energy. For a single particle moving in a potential $V(x)$, the Hamiltonian takes the form

$$\hat{H} = \hat{T} + \hat{V}(x,t) \tag{1.8}$$

where $\hat{T}$ is the kinetic energy operator, defined as

$$\hat{T} = \frac{\hat{p}^2}{2m} = -\frac{\hbar^2}{2m}\nabla^2 \tag{1.9}$$

with $\hat{p}$ being the momentum operator and m the mass of the particle and $\hat{V}(x,t)$ is the potential energy operator.

To resolve the equation, the computation of the Hamiltonian operator is first needed, then by placing it into the equation, then the resulting partial differential equation is solved for the wave function. By taking the square of the absolute value, as the Born rule says, the probability density function is obtained.

Another important form of the equation is the time-independent Schrödinger equation, used when the Hamiltonian does not explicitly depend on time. It applies to systems in stationary states, where the total energy remains constant over time. This form is useful for solving problems where the quantum state can be separated into a spatial part, describing properties like the particle's position and a time-dependent phase factor. It also describes standing waves, which oscillate in time but have a fixed amplitude profile in space.

In this case, the equation has the form

$$\hat{H}|\Psi\rangle = E|\Psi\rangle \tag{1.10}$$

where E is the energy of the system. When you measure the energy of the system in that state, the measurement will yield E every time, provided the system is in the state described by $|\Psi\rangle$. Mathematically, this is the eigenvalue corresponding to the eigenstates of the state vector. Therefore, the wave function is an eigenfunction of the Hamiltonian operator with corresponding eigenvalues E.

The time-independent Schrödinger equation is generally applied in situations such as:

- Energy eigenstates: to find stationary states with well-defined energy, where the system's energy remains constant over time.
- Bound states: describing particles confined within a region, like electrons in atoms or particles trapped in a potential well.
- Stable equilibrium: used for quantum systems in stable equilibrium, where time evolution is not taken into account.
- Separation of variables: applied when the time-dependent Schrödinger equation can be separated into spatial and temporal parts, with the spatial part governed by the time-independent equation.

1.3 Trap a Quantum Particle

Understanding how quantum mechanical principles can be used to confine a quantum particle, such as an electron, within a well-defined spatial region is of fundamental importance. A single particle is free to move in space and, relative to a Cartesian coordinate system, has three degrees of freedom (x, y, z). Trapping a particle means restricting one or more of these degrees of freedom, either partially or completely. This confinement is achieved by introducing physical boundaries or, more commonly in quantum systems, by engineering potential energy barriers. These barriers, whether electrostatic, magnetic, or structural, modify the spatial distribution of the particle's wavefunction, resulting in discrete energy states typical of quantum confinement.

The particle in a box (also called particle in a one-dimensional potential well or infinite potential well) is the simplest mathematical model illustrating how such restrictions cause energy quantization. It describes a particle confined within an idealized, one-dimensional potential well with infinitely high walls. Inside the box, the potential energy is zero, while outside it is infinite [7].

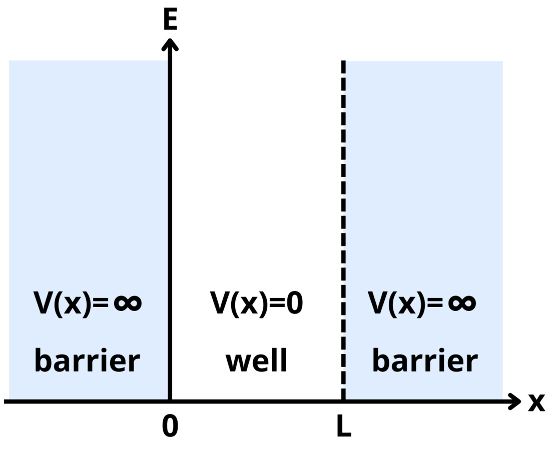


Figure 1.3: Particle in a box

The setup, shown in figure 1.3, shows that no forces act upon the particle inside the box and it can move freely in that region. However, infinitely large forces repel the particle if it touches the walls of the box, preventing it from escaping. The wave function $\psi(x,t)$ can be found by solving the Schrödinger equation for the system [12]

$$i\hbar \frac{\partial}{\partial t}\psi(x,t) = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\psi(x,t) + V(x)\psi(x,t) \tag{1.11}$$

While the particle in a box is an idealized system, it offers useful insight into how spatial confinement leads to energy quantization. In real physical systems, however, potential barriers are not infinite. This means the particle remains confined, but there is a non-zero chance of finding it just outside the well, allowing for quantum tunnelling and causing slight changes in the quantized energy levels. A more realistic model is the finite square well, where the particle faces a potential drop inside a region and finite barriers outside. In this case, the wavefunction does not drop to zero at the well's edges but instead decays exponentially into the barriers. This allows for leakage and tunnelling effects, which are important in nanoscale devices like tunnel diodes and qubit readout systems.
As in the infinite well, energy quantization still happens, but only a limited number of discrete energy levels exist within the well. These levels depend on the well's depth and width. The lowest energy states look like sine and cosine waves inside the well, smoothly connecting to decaying tails outside. The first mode is usually symmetric, while higher modes have more nodes and alternate between even and odd parity.

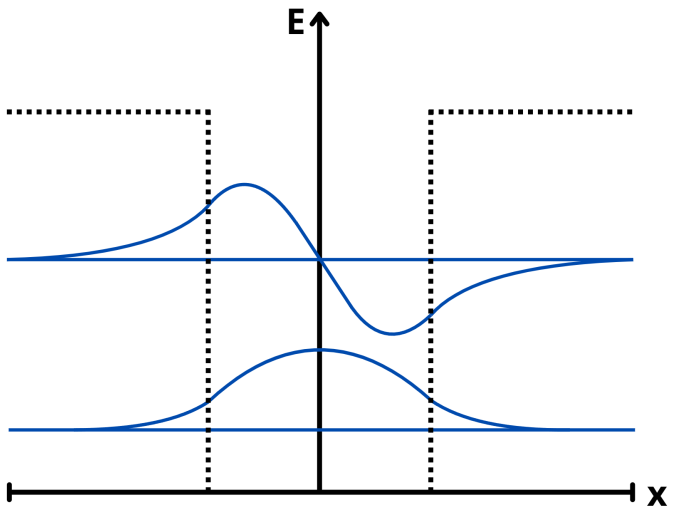


Figure 1.4: Particle in a finite potential well. First two modes.

To check for confinement, one can look at the first mode, or the first eigenfunction of the wavefunction. This corresponds to the lowest energy state in a confined system. The first eigenfunction typically represents the fundamental mode of the system, where the particle is confined within the potential well. This mode serves as an initial indicator of how the particle is localized within the position axis where the well is defined.

1.3.1 Confinement Implementation

In real devices, it is needed to trap a particle in a well defined position in three-dimensional space. If the particle looses a degree of freedom, the corresponding confinement is called quantum well, if it looses two a quantum wire and if it looses all degrees of freedom, so being trapped in a specific point where it cannot escape, it is called a quantum dot. Confinement in all spatial dimensions is achieved in quantum dots, which localize electrons and behave like artificial atoms. [26]
Confinement of quantum particles is achieved through several physical methods, each with distinct mechanisms and applications. In nanoscale devices the main mechanisms are:

Electrostatic Gates

Electrostatic gates are commonly used in semiconductor-based devices to create potential barriers that confine particles. By applying voltage to these gates, an electric field is generated, shaping the potential landscape and restricting the movement of carriers (electrons or holes) in specific regions.

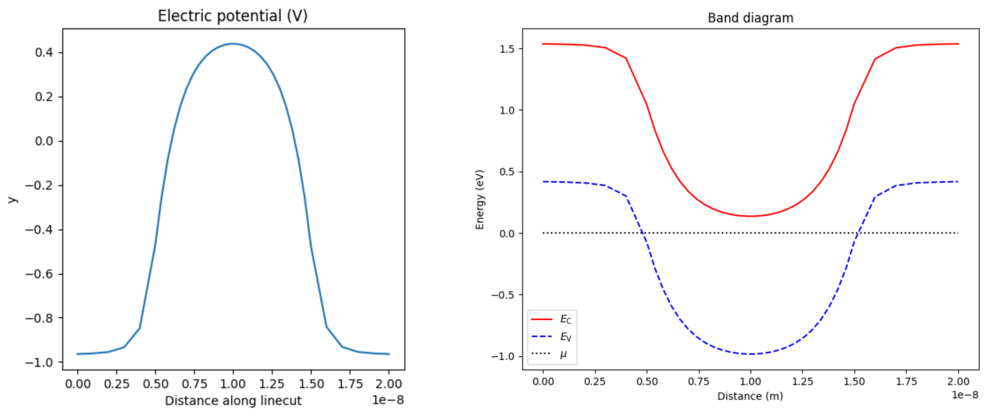


Figure 1.5: Left: Potential applied to an electrostatic gate. Right: Resulting band diagram.

Bandgap Difference

A significant technique for quantum confinement involves the use of materials with different bandgaps. This method relies on creating heterostructures where regions of different materials are layered together. The particles are confined to regions with a lower bandgap, where they cannot escape into regions with a higher bandgap, that creates a barrier for the carriers. For example, in quantum wells, the confinement occurs in two dimensions, where the particle is free to move within the well but is restricted in the perpendicular direction due to the material's bandgap contrast. This technique is usually exploited to confine a collection of non-interacting free electrons called a *gas of electrons* (GAS) in the interface between a semiconductor and a dielectric (e.g. Si/SiO_2) or two semiconductors (e.g. $Si/SiGe$, $GaAs/AlGaAs$).

Chapter 2

Spin Qubits in Solid-State Systems

Building upon the foundational principles of quantum mechanics, this second chapter introduces the concept of Qubits as the basic carriers of quantum information, focusing on spin qubits, analysing various types of qubit encoding, quantum dots and problems related to physical implementation.

2.1 Qubits Fundamentals

The term **qubit**, or **quantum bit**, coined by the American theoretical physicist Benjamin Schumacher [39], is the basic unit of quantum information and can be seen as the quantum version of the classic binary bit physically realized with a two-state device. Claude Shannon, an American mathematician known as the "father of information theory", defined the classical bit as the amount of information required to eliminate uncertainty between two equally probable and mutually exclusive outcomes [41]. In other words, one bit represents the information gained when one of two equally likely options is specified [28]. Its quantum counterpart, the qubit, is a two-state (or two-level) quantum-mechanical system used to encode such an event into a quantum of information. This refers to the smallest indivisible unit of information that can be stored in a physical property of a particle, as used in quantum computing. A qubit measurement yields one of two possible outcomes, typically labelled as "0" and "1", similar to a classical bit. However, unlike a bit, which can only be in one of these states at a time, a qubit can exist in a coherent superposition of both states simultaneously, as described by quantum mechanics [33]. The general quantum state of a qubit can be represented by a linear superposition of its two orthonormal basis states (or basis vectors). These vectors are usually denoted as: [57]

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

They are written in the conventional Dirac, or "bra–ket" notation, where the $|0\rangle$ and $|1\rangle$ are pronounced "ket 0" and "ket 1", respectively. These two orthonormal basis states,

$\{|0\rangle, |1\rangle\}$, together called the computational basis, are said to span the two-dimensional linear vector (Hilbert) space of the qubit.[58] In general, n qubits are represented by a superposition state vector in $2n$ dimensional Hilbert space.[58]

A pure qubit state is a coherent superposition of the basis states. This means that a single qubit can be described by a linear combination of ket 0 and ket 1 [11]

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

where α and β are the probability amplitudes. When a measure is performed on a qubit in the standard basis, according to the Born rule, the probability of outcome $|0\rangle$ with value "0" is $|\alpha|^2$ and the probability of outcome $|1\rangle$ with value "1" is $|\beta|^2$. Because the absolute squares of the amplitudes correspond to probabilities, α and β must satisfy the second axiom of probability theory, leading to the constraint expressed in equation 2.3 [56].

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.3}$$

The probability amplitudes, α and β, encode more than just the probabilities of the outcomes of a measurement. The relative phase between the two coefficients is, for example, responsible for quantum interference, as seen in the double-slit experiment.

The possible quantum states for a single qubit can be visualised using a **Bloch sphere**, a two-dimensional space which represents the observable state space of the pure qubit states. This state space has two local degrees of freedom, which can be represented by the two angles ϕ and θ. Since the global phase of a quantum state has no physical meaning, any overall complex factor $e^{i\varphi}$ multiplying the state can be neglected. This makes it possible to express any pure qubit state in a simplified form, where θ and ϕ are real numbers.

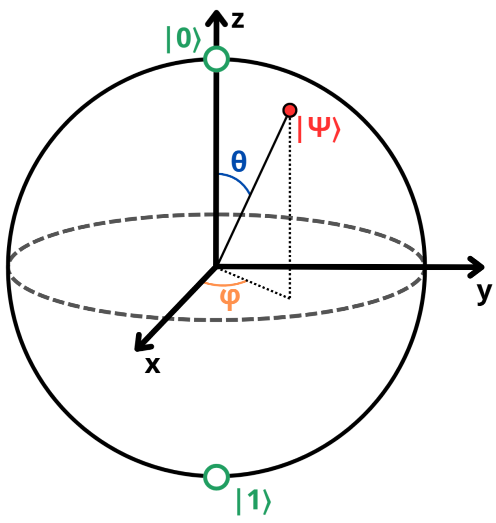


Figure 2.1: Bloch sphere representation of a qubit

Represented on a such sphere, a classical bit could only be at the "North Pole" or the "South Pole", in the locations where $|0\rangle$ and $|1\rangle$ are respectively. The rest of the surface of the Bloch sphere is inaccessible to a classical bit, but a pure qubit state can be represented by any point on the surface. For example, the pure qubit state $(|0\rangle + |1\rangle)/\sqrt{2}$ would lie on the equator of the sphere at the positive x axis. In the classical limit, a qubit, which can have quantum states anywhere on the Bloch sphere, reduces to the classical bit, which can be found only at either poles.
Here, the probability amplitudes for the superposition state are given by: [33]

$$\alpha = \cos\frac{\theta}{2} \quad \text{and} \quad \beta = e^{i\varphi}\sin\frac{\theta}{2} \tag{2.4}$$

To physically realize a qubit, it is needed to embed this mathematical concept into a 2-state system using quantum of information belonging to subatomic particles. The most common types are:

- **Superconducting Qubits:** based on electrical circuits that display quantum behavior at cryogenic temperatures. These circuits use Josephson junctions to create discrete energy levels representing quantum states. The most common design, the transmon, is optimized to minimize sensitivity to charge noise.
- **Trapped Ion Qubits:** employ individual ions (charged atoms) confined in electromagnetic traps under vacuum. Quantum information is stored in the internal electronic states of the ions and controlled using laser pulses.
- **Spin Qubits:** encode information in the spin state of single electrons or nuclei, typically trapped in quantum dots or bound to donor atoms like phosphorus in silicon. A major advantage is their compatibility with standard semiconductor processes, supporting scalability.
- **Photonic Qubits:** use photons to carry quantum information via polarization, path, or time-bin encoding. Due to their weak interaction with the environment, photons are well suited for long-distance quantum communication.
- **Topological Qubits:** rely on quasiparticles called anyons, found in certain two-dimensional systems, which follow non-Abelian statistics. Information is stored non-locally in the system's topology. In 2025, Microsoft introduced a quantum processor called *Majorana 1* [1], based on topological qubits using Majorana fermions.
- **Defect Qubits:** store quantum information in the spin states of localized electrons or nuclei linked to point defects in solid-state materials. A well-known example is the nitrogen-vacancy (NV) center in diamond or silicon carbide, where a nitrogen atom replaces a carbon atom next to a vacancy. These systems feature long coherence times, even at room temperature and can be initialized and read out optically.

2.1.1 Spin Qubits

Spin qubits have gained considerable interest in modern microelectronics thanks to their compatibility with existing semiconductor fabrication techniques. This makes it possible to integrate spin qubits into current microelectronic platforms, offering a promising path toward scalable quantum processors. As previously mentioned, spin qubits store information in the spin, an intrinsic form of angular momentum carried by elementary particles and by composite systems like hadrons, atomic nuclei and atoms. Although nuclear spin is used in the development of Donor (or Kane) qubits, where a donor atom is implanted into a semiconductor lattice, only electronic spin qubits will be described, as they are more directly relevant to the technological approaches and materials explored in this work.

In particle physics, an electron is a subatomic particle carrying a negative elementary electric charge. It belongs to the group of fermions, particles with half-integer spin that follow the Pauli exclusion principle. This principle states that two or more identical fermions cannot occupy the same quantum state simultaneously within a system governed by quantum mechanics. Austrian physicist Wolfgang Pauli formulated this principle in 1925 specifically for electrons, and it was later generalized to all fermions through his spin–statistics theorem in 1940. Because fermions obey the Pauli exclusion principle, they follow Fermi-Dirac statistics, a quantum statistical model describing how particles occupy energy states in systems where quantum effects are significant, particularly at low temperatures. This statistic gives the probability that an energy level is occupied by an electron, taking into account that no two electrons can share the same state. At absolute zero temperature, all states up to the Fermi energy are filled, while at higher temperatures there is a gradual transition, with electrons able to thermally excite to higher energy levels.

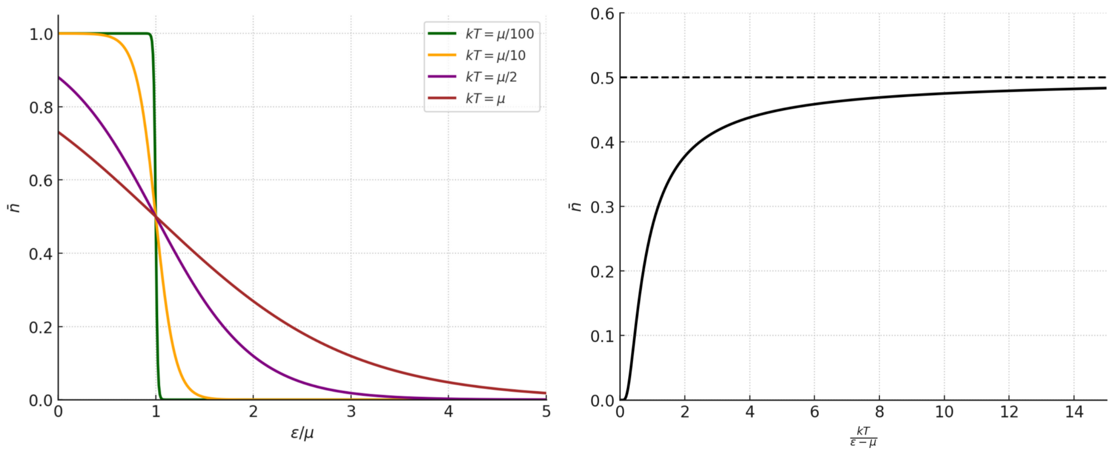


Figure 2.2: Fermi-Dirac Statistic - Energy and Temperature Dependence

In figure 2.2 it is possible to observe on the y-axis the average number of fermions in a single-particle state $\bar{n}_i$, where k is the Boltzmann constant, T is the absolute temperature, ϵ is the energy of the single-particle state i and μ is the total chemical potential.

Electrons also belongs to the Leptons family, particles that does not undergo strong interactions. [4]
As introduced before, the spin of an electron is a fundamental quantum property that can be interpreted as an intrinsic form of angular momentum. It does not correspond to any literal spinning motion of the particle, but it behaves mathematically in a way that resembles angular momentum. The spin quantum number for an electron is $s = \frac{1}{2}$, meaning that it can occupy one of two possible spin states, commonly referred to as "spin-up" and "spin-down". The concept is to exploit the two-state behaviour of the spin in order to encode the quantum information in the state up and down. These two states are denoted by the spin projections $+\frac{\hbar}{2}$ and $-\frac{\hbar}{2}$ along a chosen quantization axis, typically the z axis, such that the total angular momentum along the z axis is $S_Z = m_s \cdot \hbar$ with $m_s = \pm\frac{1}{2}$.
In simplified notation, these are labelled as:

$$+\frac{1}{2} \rightarrow \text{Spin-up} \rightarrow |\uparrow\rangle \text{ or } |0\rangle$$

$$-\frac{1}{2} \rightarrow \text{Spin-down} \rightarrow |\downarrow\rangle \text{ or } |1\rangle$$

In quantum information, the two states $|0\rangle$ and $|1\rangle$ form the computational basis of a qubit, representing its two logical states. According to quantum mechanics, an electron spin qubit can exist not only in these definite states but also in any superposition of both, collapsing to a specific state only upon measurement. Although this work focuses on the practical implementation of quantum dots for electrons, the mathematical models discussed apply to any type of charge carrier. It is therefore possible to trap and manipulate holes as well. This is commonly done in heterostructures, where holes often exhibit stronger spin-orbit coupling, an interaction between a particle's spin and its orbital motion, than electrons, especially in materials such as Germanium and III-V semiconductors. Additionally, hole spins are generally less affected by hyperfine interactions with nuclear spins, which can result in better coherence properties in certain materials.

It is also important to note that, like any quantum mechanical system, spin qubit states are subject to the no-cloning theorem, which states that it is impossible to create an independent and identical copy of an arbitrary unknown quantum state. To illustrate this in practice, consider an atom from which an electron is borrowed and trapped in a quantum dot, creating a spin qubit. When it is time to measure the spin, a process called spin-charge conversion must be performed (discussed later in the readout section). This requires the trapped electron to leave the quantum dot so its spin state can be detected, effectively reading the information while destroying the original quantum state. Afterwards, another electron can enter the quantum dot and the cycle repeats.

2.1.2 DiVincenzo Criteria

In 2000, physicist David P. DiVincenzo introduced a set of requirements, now known as the DiVincenzo criteria, to evaluate whether a physical system is suitable for building a scalable quantum computer. These criteria have become a widely accepted benchmark for evaluating different quantum computing platforms, including spin qubits in semiconductor quantum dots. Among these, semiconductor spin qubits constitute a platform that satisfies the key requirements for implementing quantum computation [5]. The original set consists of five requirements necessary for a functional quantum processor.

The five core criteria, taken from [14][15], are:

1. The elementary units of information need to be stored in a scalable quantum register. In analogy with binary logic where bits take on the value of 0 or 1, quantum information is typically stored in the form of quantum bits (qubits). A qubit is a quantum two-level system with orthogonal, i.e., distinguishable, basis states $|0\rangle$ and $|1\rangle$. Systems with spin $\frac{1}{2}$ are perhaps the simplest example of this encoding, although other spin-based possibilities exist.

2. A further requirement is that the qubits can be prepared in a fiducial state, for example, $|00..0\rangle$.

3. The quantum system must remain coherent for times much longer than the duration of elementary logic gates since decoherence causes computational errors.

4. Along with maintaining coherence, a high-fidelity gate set (single-qubit and two-qubit gates) must be attainable.

5. Finally, it is required that a sufficiently large part of the quantum register can be read out at the end of a computation.

The spin degree of freedom naturally defines a qubit, with the two states corresponding to spin-up and spin-down for a single electron [14], or alternatively to two distinct nuclear spin states [25]. Spin qubits have been demonstrated to satisfy the DiVincenzo criteria. Although the electron's charge couples strongly to electric fields, enabling electrical control of spin states, its small magnetic moment interacts only weakly with the environment, resulting in long spin coherence times [5].

2.2 Encodings

Since the spin of an electron naturally forms a two-level quantum system, it represents a convenient and intuitive way to encode a quantum bit of information. This simplest approach, where a single electron spin encodes a single qubit is conceptually straightforward and has been extensively studied both theoretically and experimentally. However, in practical implementations, spin qubits are subject to various sources of noise and decoherence, including interactions with fluctuating magnetic and electric fields, spin–orbit coupling and hyperfine interactions with nearby nuclear spins. To improve robustness, enhance gate fidelities, or enable specific operations, researchers have developed a range of alternative spin-based encodings. These schemes exploit the collective spin states of two or more particles confined in coupled quantum dots, allowing for qubit encodings that are less sensitive to certain types of noise or that facilitate more efficient control mechanisms. In the following section, the most relevant encoding strategies will be presented and discussed.

2.2.1 Loss-DiVincenzo

One of the most influential and foundational proposals for realizing spin-based quantum computation is the model developed by Daniel Loss and David P. DiVincenzo in 1997. In this approach, known as the Loss–DiVincenzo spin qubit, a single electron is confined within a quantum dot. The model is especially attractive for its compatibility with existing semiconductor fabrication technologies, making it a strong candidate for scalable architectures. Moreover, it aligns well with the DiVincenzo criteria for a physical qubit system: well-defined qubit states, initialization capability, long coherence times, universal gate implementation and qubit-specific readout. A key feature of the Loss–DiVincenzo model is its strategy for implementing two-qubit gates, which are essential for universal quantum computation. When two quantum dots are placed in close proximity, each containing a single electron, the exchange interaction between the two spins can be exploited to entangle them. This interaction, which arises from the overlap of their wavefunctions, can be dynamically controlled by adjusting the potential barrier between the dots.

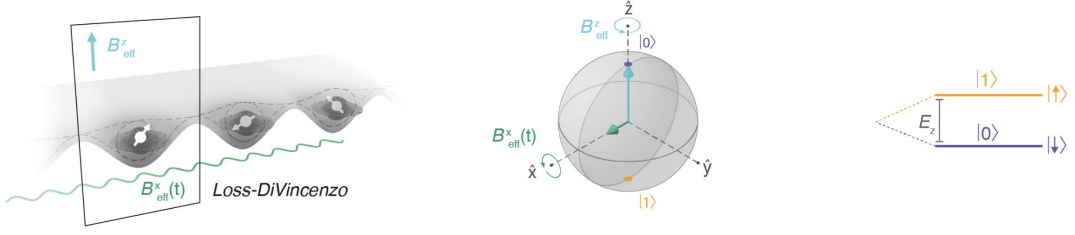


Figure 2.3: Spin configuration, Bloch sphere and energy-level diagrams associated with Loss-DiVincenzo single spin qubits [5]

2.2.2 Singlet-Triplet

Both the Loss-DiVincenzo and Kane proposals for quantum computing involve single spin qubits manipulated with a combination of static and oscillating electric and magnetic fields. The oscillating fields can be difficult to localize in nanoscale devices and the power dissipated by those fields can be problematic at cryogenic temperatures. In addition, the main source of dephasing for single spin qubits is magnetic noise from the semiconductor environment, which can be significant in materials like GaAs that contain spinful nuclei. To address these control and dephasing challenges, spin qubits can also be implemented using multispin states formed by groups of electrons. The simplest example of this approach is a qubit based on two electrons in a double quantum dot, where the controlled singlet–triplet splitting provided by the exchange interaction defines the singlet–triplet qubit [5].

The total spin configuration of two spin-½ particles spans a four-dimensional Hilbert space: [49]

$$\mathcal{H}_{2e^-} = \text{span}\{|\uparrow\uparrow\rangle, |\uparrow\downarrow\rangle, |\downarrow\uparrow\rangle, |\downarrow\downarrow\rangle\} \tag{2.5}$$

These states can be grouped into a triplet subspace with total spin $S = 1$ and a singlet state with total spin $S = 0$: [49]

- Triplet state ($S = 1$):

$$\begin{aligned} |T_+\rangle &= |\uparrow\uparrow\rangle,\ |T_-\rangle = |\downarrow\downarrow\rangle \\ |T_0\rangle &= \frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle) \end{aligned} \tag{2.6}$$

- Singlet state ($S = 0$):

$$|T_0\rangle = \frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle) \tag{2.7}$$

Encoding the qubit in the singlet–triplet basis of a double quantum dot allows a nearby charge sensor to discriminate the qubit states with high fidelity, especially at low magnetic fields. More about readout mechanisms can be found in Section 2.6.

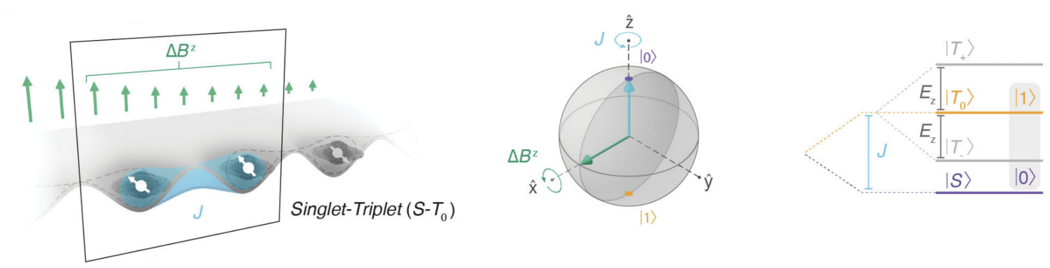


Figure 2.4: Spin configuration, Bloch sphere and energy-level diagrams associated with two-spin singlet triplet (ST0) qubits [5]

Reprinted figure with permission from Guido Burkard et al., "Semiconductor spin qubits", Rev. Mod. Phys. 95, 025003 (2023). Copyright 2023 by the American Physical Society. `http://dx.doi.org/10.1103/RevModPhys.95.025003`

2.2.3 Exchange-Only

Exchange-only is a type of spin-based qubit that encodes quantum information in the collective spin state of three (or more) electrons confined in three coupled quantum dots. Unlike single-spin or singlet–triplet qubits, which require local magnetic fields or magnetic field gradients for qubit control, the exchange-only qubit is designed to be manipulated purely by electrical means, using the exchange interaction between neighbouring electron spins.

The three-electron system spans an 8-dimensional Hilbert space, corresponding to all possible combinations of three spin-½ particles. Among these states, the total spin-½ subspace forms a natural two-level system that can be used to define a qubit. The logical basis states are encoded in the spin-½ subspace with total spin projection $S_Z = \pm\frac{1}{2}$. A commonly used basis for $S_Z = +\frac{1}{2}$ is: [37]

$$|0\rangle \equiv \frac{1}{\sqrt{2}}\left(|\uparrow\downarrow\uparrow\rangle - |\downarrow\uparrow\uparrow\rangle\right) \tag{2.8}$$

$$|1\rangle \equiv \frac{1}{\sqrt{6}}\left(2\,|\uparrow\uparrow\downarrow\rangle - |\uparrow\downarrow\uparrow\rangle - |\downarrow\uparrow\uparrow\rangle\right) \tag{2.9}$$

These states are entangled superpositions of spin configurations and are decoherence-free with respect to global magnetic field fluctuations, making them robust against certain types of noise. Applying voltages to the gates that define the potential barriers between the dots will increase or decrease the overlap between the electronic wavefunctions. This, in turn, modulates the exchange coupling, a quantum effect that favours the formation of specific spin combinations when electrons interact. For example, if the exchange interaction is turned on between the first and second electron, while the third remains weakly coupled, the system will evolve in a way that rotates the qubit state within a specific axis of its logical subspace. By then adjusting the exchange between the second and third electrons, a different rotation is achieved. In this way, any single-qubit rotation can be constructed through a sequence of controlled exchange pulses. What makes this scheme particularly attractive is that no magnetic fields are needed for manipulation.

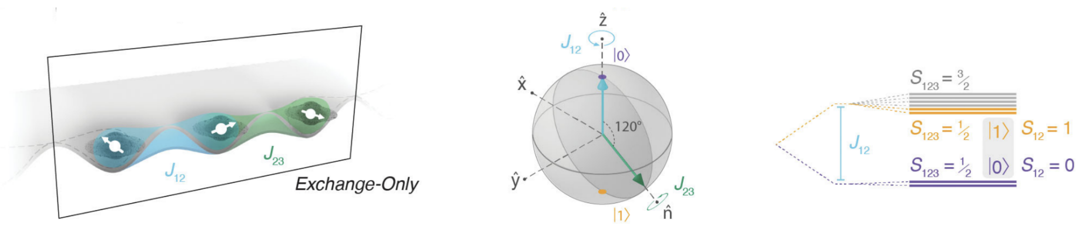


Figure 2.5: Spin configuration, Bloch sphere and energy-level diagrams associated with three-spin Exchange-Only spin qubits [5]

Reprinted figure with permission from Guido Burkard et al., "Semiconductor spin qubits", Rev. Mod. Phys. 95, 025003 (2023). Copyright 2023 by the American Physical Society. `http://dx.doi.org/10.1103/RevModPhys.95.025003`

2.3 Quantum Dots

In order to exploit the quantum mechanical effects of a particle and use it as a qubit it is firstly needed to constrain all of its degrees of freedom and trap it into a quantum dot, previously defined as a space region where the particle cannot escape. Sometimes this is called an "artificial atom" because, like real atoms, they confine electrons in all three spatial dimensions, leading to discrete, quantized energy levels. This atom-like behaviour arises from the nanoscale confinement potential, allowing quantum dots to mimic many properties of atoms, such as optical transitions, shell structure and selection rules, within a solid-state environment. This three-dimensional confinement can be exploited by electrically restrict an electron movement in space or to physically embed them in a solid material.

In order to have a quantum dot, a physical dimension called the Debye length has to be taken into account. It is a fundamental concept in semiconductor physics and electrostatics that describes the characteristic distance over which electric potentials are screened or shielded by mobile charge carriers in a material. When a local electric field is introduced, due to, for example, a charged impurity or a potential applied by a gate, the surrounding free electrons and holes will rearrange themselves to counteract the field, reducing its effect at larger distances. It represents the scale at which the electrostatic potential drops by a factor of e^{-1}. In practice, it determines how far a perturbation in charge or potential can influence the surrounding medium.
The computation that is usually done consists in making the thermal energy smaller than the energy of the first energy level of a quantum well: [34]

$$k_B T < \frac{\hbar^2 \pi^2}{2 m_{\text{eff}} L^2} \qquad L_{QD} < \lambda_{Debye} = \frac{h}{\sqrt{8 m_{\text{eff}} k_B T}} \tag{2.10}$$

Quantum confinement occurs when the dimensions of a system, such as a quantum dot, become comparable to or smaller than the characteristic length scales of the carriers, such as their de Broglie wavelength. In this regime, the energy levels of electrons or holes become discrete and their behaviour must be described by quantum mechanics rather than classical physics. The Debye length enters the picture as it defines how far electrostatic potentials, such as those from gates or charged impurities, can influence the carrier distribution in a semiconductor. If the Debye length is longer than or comparable to the size of the quantum dot, the external electrostatic environment can significantly affect the confinement potential, tuning the energy levels and wavefunctions of the carriers inside the dot. Conversely, in highly doped materials, where the Debye length becomes very short, the ability of external gates to penetrate and modulate the potential landscape is reduced, making it harder to achieve precise quantum confinement using electrostatic control.
If the case of silicon is considered, it can be noticed that at room temperature confinement would manifest only for dimensions below few nm. This means that in conventional CMOS devices, even if the 2DEG is closed to the Si/SiO_2 interface, confinement never happens and it starts playing a role only in the very last technological nodes. On the other hand, in materials like $GaAs$ and $InAs$ confinement begins for larger dimensions with respect

to silicon at the same temperature. This is due to the fact that electrons in such materials have a lower effective mass, and so a higher mobility. However, if cooled down to $4.2K$ (the boiling point of He^4), also in silicon quantum confinement starts manifesting in wells smaller than about 60–80 nm. For this reason, quantum dots have a typical size of 50 nm or less. [35]

There are various mathematical forms of the equation 2.10, by using the one shown in 2.11 [9]

$$\lambda_D = \sqrt{\frac{\varepsilon k_B T}{ne^2}} \tag{2.11}$$

and plotting by using permittivity of various materials at a carrier concentration of $1e20m^{-3}$ it is possible to see, in figure 2.6, the Debye length of various semiconductors varying the temperature from absolute zero to room temperature.

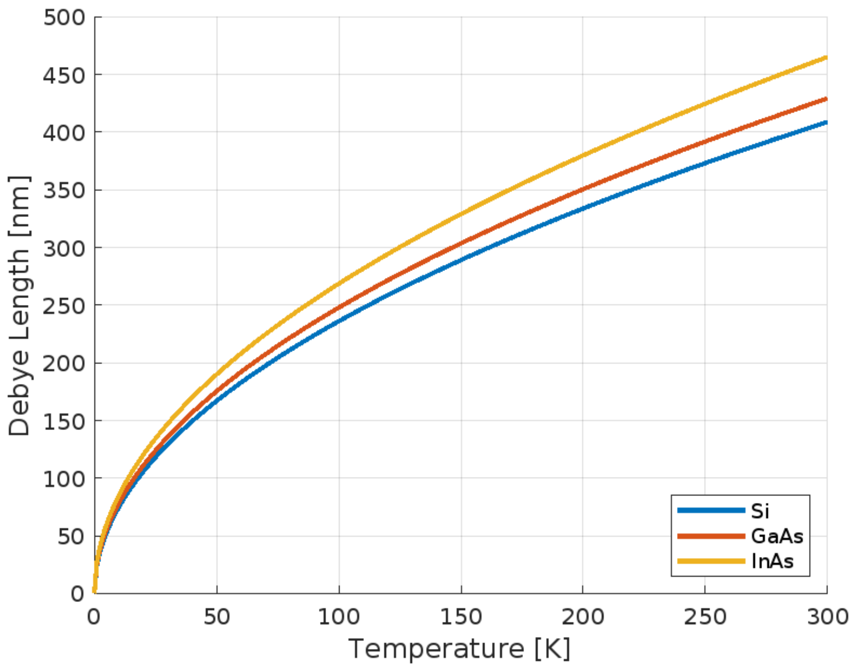


Figure 2.6: Debye length of Silicon, Gallium Arsenide and Indium Arsenide, varying the temperature from absolute zero to room temperature, carrier concentration $1e20m^{-3}$

It is possible to observe that at absolute zero the particle, completely drained of its total energy, cannot escape the quantum dot. The Debye length it is not the only quantum mechanical effect to respect, many other phenomenons will arise with the increase of just few Kelvin. This is the reason why even modern technological nodes still tend to be operated in a sub-kelvin regime.

2.3.1 Gate Defined Quantum Dots

Trapping single spins requires quantum confinement, which is usually achieved through a combination of material-defined and electrostatically defined spatial barriers [5]. Gate-defined quantum dots are nanostructures formed in semiconductor materials through the application of electrostatic potentials generated by metallic gates. The aim of these gates is to reduce one or two degrees of freedom, usually x and/or y axis. The gates are patterned on top of a semiconductor stack, typically a silicon-based MOS structure or a GaAs/AlGaAs heterostructure, where a two-dimensional electron gas (2DEG) or hole gas (2DHG) is present at the interface, confining the carriers in the z axis. By applying voltages to selected gate electrodes, the underlying carriers are locally depleted, creating potential wells that confine a small number of charge carriers in all three spatial dimensions. So, in order to obtain a gate-defined quantum dot, it is necessary to engineer:

Band structure (quantum well formation)

The first step is to reduce the carriers degrees of freedom by creating a quantum well. This is done by exploiting the band structure of the different materials that compose the device. It is possible to obtain, mainly, two types of material confinement:

- Semiconductor/oxide interface: as in MOS devices, the gas of electrons (or holes) is present at the interface semiconductor/oxide, due to the bandgap difference of the semiconductor and the dielectric. This is common in depletion-mode devices such as Si-MOS and compatible with standard processes.
- Heterostructures: buried few nanometers below the semiconductor surface, an additional layer of a different semiconductor is present. The most common types are $SiGe$ for the main semiconductor and Si or Ge for the buried layer. The different bandgap of the two semiconductors lead to a confinement of carriers inside the thin buried layer, forming a 2DEG (or 2DHG).

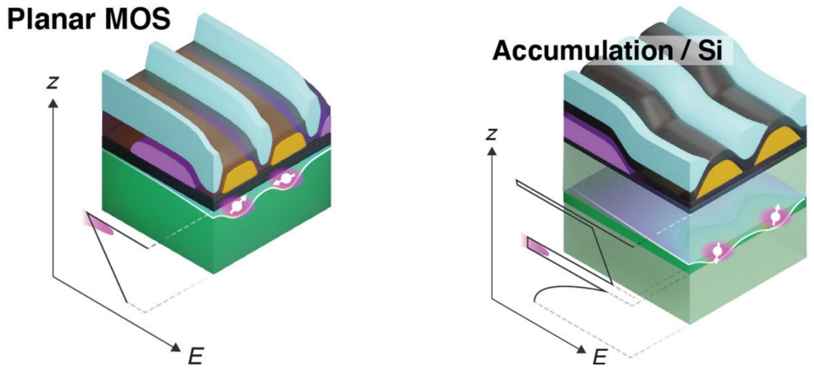


Figure 2.7: Physical structure with band diagram. In Si-MOS, electrons are localized at the Si/SiO_2 interface. In heterostructures, the electrons reside in a buried QW [5]

Reprinted figure with permission from Guido Burkard et al., "Semiconductor spin qubits", Rev. Mod. Phys. 95, 025003 (2023). Copyright 2023 by the American Physical Society. http://dx.doi.org/10.1103/RevModPhys.95.025003

Electrostatic gating (quantum dot formation)

Once a quantum well has been formed in a planar heterostructure, confinement in the in-plane dimensions can further reduce the effective dimensionality of the electronic states. In-plane confinement is achieved through the electrostatic potential, which is typically induced by metal gate electrodes above the heterostructure. A confining potential along one direction creates a quasi-1D channel, which can serve as a quantum point contact. Applying finer electrostatic confinement along both in-plane directions can produce effectively zero-dimensional quantum dots. The potential minima determine the locations of the quantum dots where electrons can be trapped and variations in gate voltage modify both the electrochemical potential of the dots and the shape of the confining potential [5].

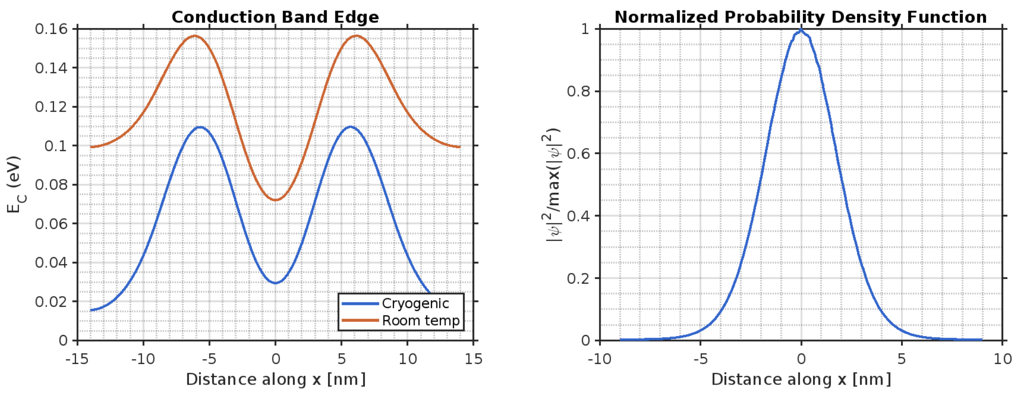


Figure 2.8: Left: Band diagram of a gate-defined quantum dot, the central minima represent the dot, the two lateral walls represent a potential barrier that traps the carriers in the potential minima. Right: normalized probability density function of the carriers along the axis in which the channel extends

The simplest structure used to confine a carrier along one direction implies three gates placed one after the other. The central one is called plunger gate and it is used to attract the 2DEG to the interface semiconductor-oxide. The two gates at the side of the plunger are the barrier gates, which are inversely polarized with respect to the plunger, in order to create a potential barrier where carries cannot enter nor escape without tuning the voltages. The behaviour described in figure 2.8 are directly taken from a quantum simulation of such a device. Here the two barrier gates have a low-magnitude negative potential ($\sim -0.1V$) that lead to the formation of the potential barriers, while the plunger gate have a positive potential ($\sim +0.6V$) that flatten the bands in the central region, creating the dot. By finely tuning the voltages applied to the plunger and barrier gates, it is possible to control the number of confined electrons with single-electron precision, a key requirement for quantum information processing.

Moreover, by adding additional sets of gates, multiple quantum dots can be defined in close proximity, enabling the study of tunnel coupling, charge hybridization and coherent exchange interactions. These coupled quantum dot systems, often referred to as double quantum dots (DQDs) or linear arrays, are essential building blocks for scalable qubit architectures. The ability to electrostatically manipulate confinement, tunnelling and inter-dot coupling makes gate-defined quantum dots a versatile and widely studied platform for the implementation of spin- and charge-based qubits in solid-state systems.

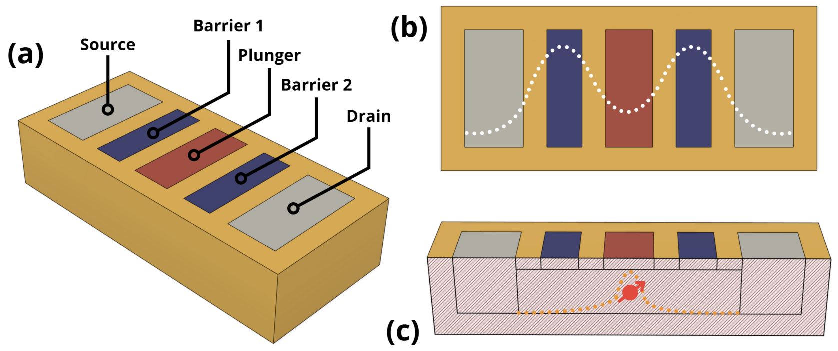


Figure 2.9: (a) Example of a planar device on SOI wafer with a two-barrier gates (in blue) and one-plunger gate (in red) configuration. The grey areas are the n++ source/drain areas and the orange areas represent the oxide, both lateral and buried. (b) Top view with applied potential (white dotted line). (c) Side cut with probability density function $|\psi(x)|^2$ computed in the silicon channel (orange dotted line) and spin qubit location (marked in red)

In a double quantum dot, the interdot barrier height can be voltage controlled to modulate the interdot tunnel coupling. Typical devices employ separate plunger and barrier gates to control the quantum dot electrochemical potentials and the interdot barriers, respectively. In practice, geometrical cross-capacitances affect the potential under neighbouring gates, so voltage compensation across multiple gates is needed to control each dot independently, a process often referred to as defining “virtual gates” [5].

2.3.2 Dopants

In addition to gate-defined quantum dots, where electrostatic potentials are used to confine electrons in a two-dimensional electron gas, an alternative and intrinsically atomic-scale approach involves the use of dopant atoms embedded in a semiconductor crystal. These so-called dopant-based quantum dots exploit the localized potential wells created by impurity atoms, typically donors such as phosphorus in silicon, which can trap single electrons in discrete, hydrogen-like bound states. When a group-V donor atom (e.g., phosphorus, arsenic or antimony) is introduced into a silicon lattice, it donates a loosely bound electron that becomes confined in a Coulombic potential near the dopant nucleus. At low temperatures and in the presence of appropriate gate control, the donor-bound electron behaves similarly to an electron confined in a quantum dot, with discrete energy levels and a well-defined spin state. This system forms a natural candidate for hosting a spin qubit. Because the confinement is provided by the atomic potential of the dopant itself, these structures are extremely small and reproducible at the atomic scale, offering excellent charge and spin stability. Additionally, the nuclear spin of the donor atom can serve as a second, highly coherent quantum degree of freedom, enabling hybrid electron–nuclear spin qubit architectures.
The integration of dopants into silicon using standard ion implantation or scanning probe lithography techniques also provides compatibility with conventional CMOS fabrication, making dopant-based systems promising for scalable quantum device architectures.

Kane's Qubit

Among all implementations, the most notable and historically significant dopant-based spin qubit is the Kane qubit, proposed in 1998 by Bruce E. Kane. Soon after the Loss-DiVincenzo proposal on quantum computation using quantum dots, Kane suggested using the nuclear spins of ^{31}P donor atoms in silicon to build a quantum computer [25]. Nuclear spins are highly coherent since the nuclear gyromagnetic ratio for ^{31}P is nearly 2000 times smaller than the electron gyromagnetic ratio and their lack of mobility in a solid state host inhibits charge-hybridizing or spin-orbit-related decoherence mechanisms. Control over individual qubits is achieved through a combination of static magnetic fields and voltage-controlled gates. Specifically, "A-gates" positioned above each donor atom modulate the hyperfine interaction between the donor's nuclear spin and its bound electron. By adjusting the voltage on these gates, the resonance frequency of individual nuclear spins can be tuned, allowing for selective addressing and manipulation using radio frequency pulses. Two-qubit operations are facilitated by "J-gates" located between adjacent donor atoms. These gates control the exchange interaction between neighbouring donor electrons. By temporarily transferring the quantum information from the nuclear spin to the electron spin and then enabling interaction between electrons of adjacent donors, entangling operations can be performed.

The Kane model's reliance on the nuclear spin's long coherence time, combined with the scalability potential of silicon-based fabrication techniques, makes it a compelling approach within the Loss–DiVincenzo paradigm. However, challenges like the precise placement of donor atoms and the control of individual qubits remain active areas of research. Kane himself noted in 1998 the extreme difficulty of fabricating devices at the single-atom level, yet he argued that the semiconductor industry's progress in miniaturization driven by Moore's law would eventually enable the production of silicon devices at the scale required for quantum computing [6].

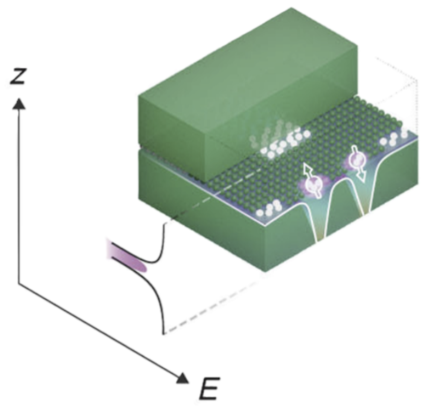


Figure 2.10: Donor electrons are confined by the positive potential of the donor atom and manipulated with gates defined through conventional or STM lithography [5]

Reprinted figure with permission from Guido Burkard et al., "Semiconductor spin qubits", Rev. Mod. Phys. 95, 025003 (2023). Copyright 2023 by the American Physical Society. `http://dx.doi.org/10.1103/RevModPhys.95.025003`

2.3.3 Nitrogen-Vacancy Centers

Nitrogen-Vacancy (NV) centers are point defects in crystalline materials that behave similarly to quantum dots, providing discrete, atom-like energy levels within a solid-state environment. They are among the most studied solid-state systems for quantum information processing and sensing due to their unique combination of quantum coherence, optical addressability and room-temperature operation. An NV center is formed when a nitrogen atom substitutes for a carbon atom in the diamond lattice, adjacent to a vacant site where a second carbon atom is missing. This defect forms a localized electronic structure that traps electrons and exhibits quantized energy levels similar to those of an artificial atom. In its negatively charged state NV^-, the center possesses an electronic spin triplet ground state, which can be used to encode quantum information. The spin sublevels of the NV^- ground state can be selectively manipulated using microwave fields, and their population can be initialized and read out optically, thanks to a spin-dependent fluorescence response.

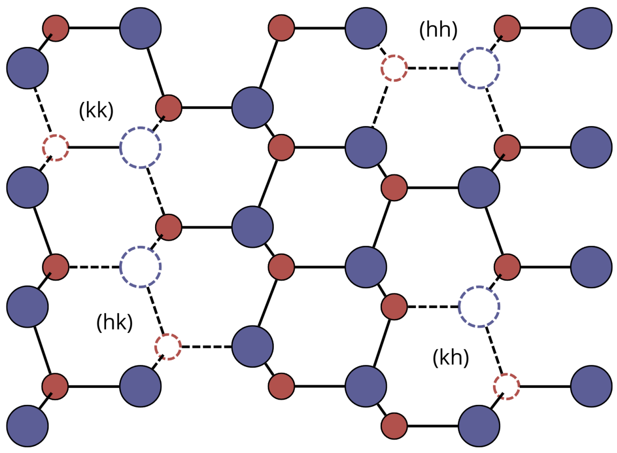


Figure 2.11: Atomic structure of silicon carbide, highlighting divacancy lattice defects. Dotted blue and red circles indicate individual silicon and carbon vacancies. Four distinct configurations of these vacancies can occur, depending on whether they occupy h or k lattice sites [30]

The electron and nuclear spins associated with nitrogen–vacancy centers in diamond and phosphorus impurities in silicon are among the most coherent quantum systems that have been individually observed and manipulated in the solid state. These two materials offer complementary advantages for potential spintronic applications: diamond provides optically accessible quantum states and supports room-temperature operation, while silicon excels in fabrication and electrical interfacing. Silicon carbide, as a compound of these two materials, could potentially combine the advantages of both, enabling a single optically addressable spin with long coherence times at room temperature, embedded within a high-performance electronic platform [30].

2.4 Noise, Decoherence and Error

Spin qubits are typically operated at cryogenic temperatures, often below $100mK$, using dilution helium refrigerators. These low temperatures serve two primary purposes. First, they suppress thermal excitations that can randomly flip qubit states, ensuring that qubits remain in their ground state and enabling initialization with high fidelity. Second, cryogenic conditions reduce the coupling of the qubit to environmental noise sources, thereby extending coherence times. Spin qubits are not immune to noise and suffer from a phenomena called decoherence, the loss of quantum coherence due to interactions with the environment. As long as it maintain correctly the spin state, we call it in a coherent state, while after some time it became decoherent with respect to what it should be and so the quantum information is lost and so it is necessary to fill the dot again. This phenomena limit the fidelity of quantum operations and are a major challenge for scalable quantum computation. Understanding and mitigating these effects is critical for building reliable quantum systems.
Two primary timescales characterize decoherence in spin qubits:

- **Relaxation Time** T_1: This is the time over which a qubit loses energy to its environment, transitioning from an excited state to its ground state. For electron spin qubits in silicon quantum dots, T_1 times can range from microseconds to seconds, depending on the specific system and environmental conditions.
- **Dephasing Time** T_2: This is the time over which a qubit loses phase coherence without necessarily losing energy. T_2 is often shorter than T_1 and is influenced by various dephasing mechanisms, including interactions with nuclear spins and charge noise.

In many spin qubit systems, T_1 is significantly longer than T_2, indicating that dephasing is the dominant decoherence mechanism. "For instance, in a silicon quantum dot system, T_1 has been measured at $280\mu s$, while T_2 is around $250\mu s$ at $0.35K$."[21] So, it is crucial to end quantum computation with qubits before the decoherence time.

Several factors contribute to noise and decoherence in electron spin qubits:

- Environmental Noise: Fluctuating electric and magnetic fields from the environment can couple to the qubit, causing dephasing or even bit flips. Magnetic noise can be particularly harmful for spin qubits, but artificial controlled fields can be used to flip the qubit as needed (more in Section 2.6).
- Hyperfine Interactions: In natural silicon or other host materials, the presence of nuclear spins creates a fluctuating magnetic background, known as the hyperfine interaction. This is a significant source of decoherence. Using isotopically purified silicon reduces nuclear spin noise drastically, thereby improving coherence times (more in Section 3.1).
- Material Imperfections: Defects, charge traps and interface roughness at the 2DEG boundaries introduce local noise sources and variability in the qubit environment.

2.5 Transport Analysis

The term "transport" refers to the movement of electrons through the quantum dot system when a voltage is applied between the source and drain electrodes. Thus, while qubit operations themselves are often isolated from transport, the transport regime is essential during initialization, tuning and readout phases of the device. When a quantum dot is weakly coupled to source and drain electrodes via tunnel barriers and a gate electrode controls its electrochemical potential, it can exhibit a phenomenon known as Coulomb blockade. The simplest device in which the effect of Coulomb blockade can be observed is the so-called single-electron transistor (SET). It consists of two electrodes known as the drain and the source, connected through tunnel junctions to one common electrode with a low self-capacitance, known as the island. The electrical potential of the island can be tuned by a third electrode, known as the gate, which is capacitively coupled to the island.

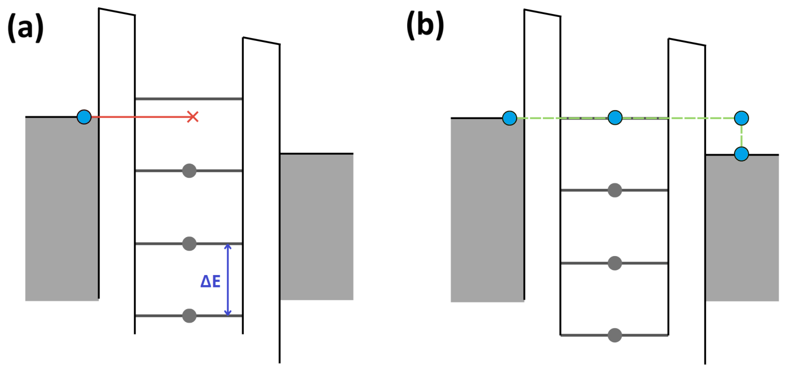


Figure 2.12: Coulomb blockade example, energy levels of source, island and drain in a single-electron transistor. The energy levels of the island electrode are evenly spaced with a separation of ΔE. (a) Blocking state. (b) Transmitting state.

In this configuration, electrons tunnel from the source to the quantum dot, causing a transition from N to $N+1$ electrons within the dot, followed by tunnelling toward the drain. This sequence generates fluctuations in the electron number inside the dot, producing measurable conductance peaks, referred to as Coulomb peaks. These peaks appear at specific gate voltages that allow the dot to be filled with electrons one by one [42]. In the Coulomb blockade regime, the addition of a single electron to the dot requires overcoming a charging energy due to electron–electron repulsion. As a result, electron transport through the dot only occurs when the energy levels align appropriately with the chemical potentials of the source and drain leads.
Another important graph is the so called charge stability diagram, that for a single quantum dot shows the so called Coulomb diamond. It is a 2D map showing the differential conductance (or current) as a function of source-drain bias voltage V_{SD} (vertical axis) and gate voltage V_G (horizontal axis). It typically displays diamond-shaped regions of zero conductance, the so-called Coulomb diamonds, each corresponding to a fixed number of electrons in the quantum dot. Inside a diamond the dot is in Coulomb blockade and

current is suppressed. At the diamond edges transport is allowed due to alignment of dot energy levels with the source or drain. The half height of a diamond relates to the charging energy E_C, which is the energy required to add an extra electron to the dot.

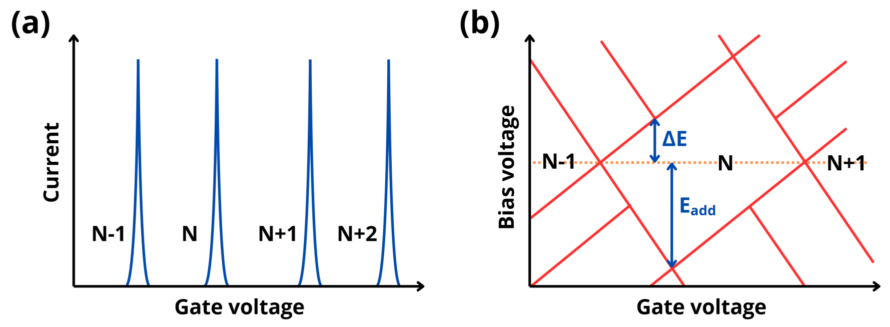


Figure 2.13: Transport through a quantum dot. (a) Coulomb peaks in current versus gate voltage in the linear-response regime. (b) Coulomb diamonds in differential conductance versus V_{DS} and V_g

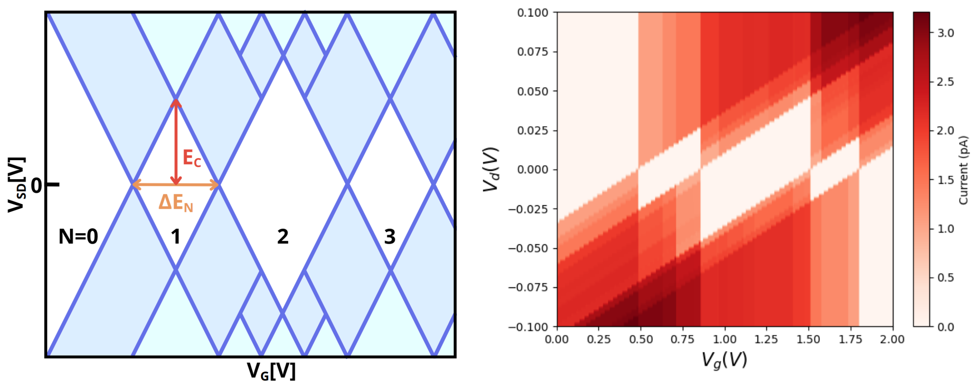


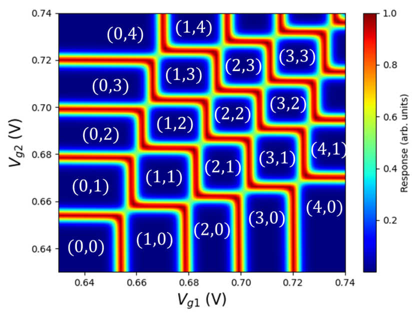
Figure 2.14: Left: A graphic showing the distance-height (V_G, V_{SD}) relations between diamonds. Right: simulated Coulomb diamonds.

The charging energy E_C is given approximately by: [29]

$$E_C = \frac{e^2}{C_\Sigma} \gg k_B T \tag{2.12}$$

where e is the elementary charge and C_Σ is the total capacitance of the quantum dot with respect to its environment (including source, drain and gate electrodes). A taller diamond therefore indicates a larger E_C, this typically implies that the capacitance C_Σ is smaller. Consequently, the physical size of the quantum dot is smaller, since capacitance scales with geometry. In general, a taller Coulomb diamond indicates stronger confinement and smaller quantum dot size, which is generally advantageous for the stable operation of a spin qubit.

To interact with the qubit, it is usually coupled to another one in a double quantum dot configuration. This configuration allow for interfacing via a quantum wire or SET (more on section 2.6). Again, the charge stability diagram is a fundamental tool for characterizing and operating double quantum dot systems. It represents the stable charge configurations of the double quantum dot, usually measured by recording the current or differential conductance of the system as a function of the two plunger gate voltages, which control the chemical potential of each dot [31], typically denoted as V_{G1} and V_{G2}. As it denotes the carrier occupancies in a double quantum dot configuration, it is commonly referred as the particle addition spectrum.
The plane defined by V_{G1} and V_{G2} is partitioned into regions where the number of electrons on each dot remains constant. These regions often form a characteristic honeycomb pattern, with each cell labelled by a pair of integers (N_1, N_2), representing the electron count on dot 1 and dot 2, respectively. The boundaries between these regions correspond to transitions where the addition or removal of an electron becomes energetically favourable, leading to changes in the charge state of the system. At points where three regions meet, known as triple points, the system can fluctuate between three different charge states. These points are of particular interest because they allow for electron tunnelling between the dots and the leads, facilitating transport measurements that are sensitive to the quantum properties of the system.



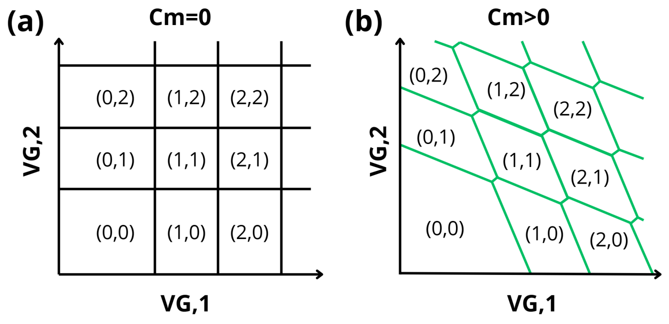


Figure 2.15: Left: Particle addition spectrum with labelled dot occupations. [31] Right: Charge stability diagram of double quantum dots acquired by scanning V_{G1}, V_{G2}. (a) Uncoupled and (b) coupled quantum dots can be transformed reversibly into each other by tuning C_m. C_m is the cross capacitance between the dots. The lines in the diagram are the current signals while sweeping the voltage.[54][51]

When two quantum dots are strongly coupled, either capacitively or via tunnel coupling, the charge stability diagram deviates from the idealized honeycomb pattern observed in weakly coupled systems. In this case, the lines that separate different charge states begin to bend, indicating that the dots are no longer isolated electrostatically but are forming a coherent quantum system. This is noticeable in figure 2.15 (right, (b)).

2.6 Qubit Control

The operation of spin qubits relies on three fundamental steps: initialization, manipulation and readout. Each of these stages can be implemented in multiple ways depending on the physical qubit architecture and the surrounding electronics. Here, the most common methods used to perform these operations in spin-based quantum dot systems are reviewed.

2.6.1 Initialization

The initialization of spin qubits is a crucial step in quantum computation, as it prepares the system in a well-defined quantum state from which controlled operations can be reliably performed. For electron spin qubits hosted in semiconductor quantum dots, the goal is typically to initialize the system into a known spin state, usually the spin-down (or spin-up) state depending on the applied magnetic field direction. Initialization is often achieved via energy-selective tunnelling, relaxation to the ground state, or adiabatic preparation. The choice of initialization technique depends on the specific qubit encoding and system architecture.

In the Loss-DiVincenzo single-spin encoding, initialization is typically achieved by tuning the quantum dot so that an electron can tunnel in from a reservoir only if its spin matches the ground state, determined by the Zeeman splitting in an external magnetic field. While the original proposal suggested using spin-selective ferromagnetic elements, practical implementations rely on spin-selective tunnelling to a fermionic electron bath. A large static magnetic field $B \gg k_B T_e = g\mu_B$ ensures that the higher-energy spin state can tunnel to the Fermi sea, while tunnelling from the lower-energy state is energetically forbidden. Sensitive charge detectors measure the presence or absence of tunnelling events, allowing the electron spin orientation to be inferred. This spin-selective tunnelling enables high-fidelity initialization of the qubit into a well-defined spin state [5].
For Singlet–Triplet qubits, which encode information in the two-electron spin states within a double quantum dot, initialization is typically performed by biasing the system into the (0,2) charge configuration, where both electrons occupy the same dot and relax to a spin-singlet ground state due to the Pauli exclusion principle and exchange interaction. The system can then be adiabatically brought to the (1,1) configuration, preserving the singlet spin character.
In the case of Exchange-only qubits, a common strategy is to first prepare a singlet state in one of the dot pairs, then load the third electron and manipulate the exchange couplings so that the system evolves into the logical $|0\rangle$ state within the encoded basis. This often involves careful pulse sequences and gate voltage tuning.

Overall, reliable initialization is essential for achieving high-fidelity quantum operations and ongoing research continues to optimize these techniques for speed, scalability and compatibility with cryogenic control electronics.

2.6.2 Manipulation

Qubit manipulation can be carried out in many ways, depending on the type of qubit and quantum dot implementation. The main idea is to coherently controlling their quantum state by inducing transitions between the basis states $|0\rangle$ and $|1\rangle$, like in classical bits.
The main techniques involve applying resonant microwave pulses tuned to the qubit's Larmor frequency [6], the rate at which a spin precesses around an external static magnetic field due to the Zeeman interaction, which is defined as: [5]

$$f_L = \frac{g\mu_B B}{h} \tag{2.13}$$

where g is the Landé g-factor, μ_B is the Bohr magneton, B is the static magnetic field and h is Planck's constant. The Larmor frequency sets the energy difference between the spin states and determines the frequency of the driving field required for coherent control. When a spin qubit is driven by such a resonant oscillating field, it undergoes Rabi oscillations, coherent oscillations of a qubit's state population that occur when it is driven by an external oscillating field at or near the Larmor frequency. The probability of finding the spin in a particular state oscillates sinusoidally as a function of the duration of the driving pulse. The oscillation frequency, known as the Rabi frequency, depends on the strength of the driving field and determines the speed of spin rotations on the Bloch sphere. The duration and amplitude of the control pulse allow for precise rotations of the qubit state on the Bloch sphere.

Electron Spin Resonance (ESR)

Electron Spin Resonance is one of the most fundamental techniques for manipulating spin qubits. It involves applying an oscillating magnetic field perpendicular to a static external magnetic field to induce coherent transitions between the spin-up and spin-down states. A static magnetic field B_0 is first applied to lift the degeneracy of the spin states via the Zeeman effect, creating an energy splitting: [5]

$$\Delta E = g\mu_B B_0 \tag{2.14}$$

To manipulate the spin, an alternating magnetic field B_1, typically in the microwave frequency range, is applied at the Larmor frequency corresponding to the Zeeman splitting (see equation 2.13). When the microwave field is on resonance with this frequency, the spin undergoes Rabi oscillations, enabling arbitrary single-qubit rotations by controlling the duration and amplitude of the microwave pulse.
ESR is primarily used to control single-spin qubits, such as those described by the Loss–DiVincenzo encoding, where a qubit is defined by the spin state of a single electron confined in a quantum dot. Because ESR addresses individual spins using magnetic resonance, it is best suited for architectures where each spin qubit can be isolated and selectively driven, such as single quantum dots with local microwave antennas or silicon-based devices where spin coherence times are relatively long. ESR is not typically used for multi-spin encodings like singlet–triplet or exchange-only qubits, which rely on exchange interactions and electric control rather than magnetic resonance for manipulation.

Electric Dipole Spin Resonance (EDSR)

Electric Dipole Spin Resonance is a technique used to manipulate spin qubits by applying an oscillating electric field, rather than a magnetic field, to induce spin rotations. This method exploits the coupling between the electron's spin and its orbital motion through mechanisms such as spin–orbit interaction or an artificial magnetic field gradient. In systems with sufficient spin–orbit coupling (such as III–V semiconductors like InAs or GaAs), an electric field can induce an effective magnetic field in the rest frame of the electron, allowing transitions between spin states when the field is applied at the Larmor frequency. In silicon-based quantum dots, where intrinsic spin–orbit coupling is weaker, EDSR is often enabled by introducing a static magnetic field gradient, typically using a nearby micromagnet. As the electron is displaced by the oscillating electric field, it experiences a time-varying magnetic field due to the gradient, which drives spin rotations. EDSR is especially attractive for scalable quantum computing architectures because electric fields are easier to generate, confine and integrate than magnetic fields.
Like ESR, EDSR is used with single-spin qubits and generally not for multi-spin qubit encodings like singlet–triplet or exchange-only ones, which rely on direct control of exchange interactions instead.

Exchange-Based Gates

Exchange-based gates are quantum operations that manipulate spin qubits by controlling the exchange interaction, a quantum mechanical effect that couples the spins of two nearby electrons. Unlike ESR or EDSR, which require oscillating magnetic or electric fields to induce spin rotations, exchange-based gates rely purely on electrostatic control of gate voltages that modulate the tunnel coupling between quantum dots.
When two electrons are placed in adjacent quantum dots and their wavefunctions overlap, an exchange interaction J arises, described by the Heisenberg Hamiltonian: [5]

$$H = J(t)\vec{S}_1 \cdot \vec{S}_2 \tag{2.15}$$

Here $\vec{S}_1$ and $\vec{S}_2$ are the spin operators for the two electrons and $J(t)$ is a time-dependent exchange coupling that can be tuned by adjusting gate voltages. This interaction causes the two-spin state to evolve coherently, enabling entangling gates (such as the $\sqrt{\text{SWAP}}$ or full SWAP) and, with proper encoding, single-qubit rotations as well.

Exchange-based gates are essential for controlling qubit encodings that use multiple spins, such as singlet–triplet qubits, where the logical qubit is defined by the spin configuration of two electrons in a double quantum dot and exchange-only qubits, where three electron spins are used to define a logical qubit and all operations are performed using only the exchange interaction, without requiring spin resonance techniques.

2.6.3 Readout

Accurate and efficient readout of spin qubit states is a crucial step in any quantum computing protocol. The readout process involves measuring the spin state of the qubit and converting this quantum information into a classical signal that can be processed and analyzed. Due to the fragile nature of spin states and their susceptibility to decoherence, readout techniques must be both highly sensitive and minimally invasive to preserve qubit coherence for subsequent operations. A key principle underlying most spin qubit readout methods is spin-to-charge conversion. Because direct measurement of the electron spin is challenging, this technique translates the spin information into a detectable charge state change. This conversion enables the use of sensitive charge sensors to perform high-fidelity spin state measurements indirectly.

Elzerman Method

The Elzerman method is a widely used technique for reading out the spin state of a single electron confined in a quantum dot, firstly proposed by J. M. Elzerman in 2004 and therefore it carries his name [17]. It exploits spin-dependent tunnelling between the quantum dot and a nearby electron reservoir under the influence of a carefully tuned magnetic field and electrostatic potential. Due to the Zeeman splitting induced by an external magnetic field, the spin-up and spin-down states have different energies. By adjusting the quantum dot's energy levels relative to the reservoir's Fermi level, it is possible to selectively allow an electron with one spin orientation (typically spin-up) to tunnel out of the dot, while the other spin state remains trapped.
During the readout process:

- If the electron is in the spin state allowed to tunnel out (spin-up or $|0\rangle$), it leaves the dot, creating a temporary change in the dot's charge state. This charge change is detected by a nearby charge sensor, such as a quantum point contact or a single-electron transistor.
- If the electron spin state is not allowed to tunnel out (spin-down or $|1\rangle$), it remains in the dot, no charge change is detected.

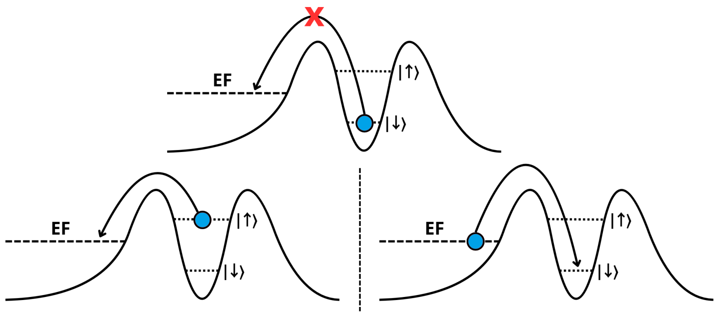


Figure 2.16: Elzerman readout configuration

By monitoring the charge sensor signal in time, the spin state can be inferred with high fidelity. The Elzerman method effectively converts the spin information into a measurable charge signal, leveraging spin-to-charge conversion. This method requires relatively long spin relaxation times (so the electron does not flip spin spontaneously during readout) and precise tuning of the dot's energy levels, but it provides a robust and accessible way to perform single-shot spin readout.

Pauli Spin Blockade Method

The Elzerman technique provides the simplest method for readout, but it has a significant limitation: an electron reservoir must be accessible near each quantum dot to perform the measurement. [13] This matter is solved through the Pauli Spin Blockade, a readout mechanism used primarily in double quantum dot systems to distinguish between different spin states of two electrons. It exploits the Pauli exclusion principle, which forbids two electrons with the same spin state from occupying the same quantum state. It removes the need of reservoirs and instead makes use of ancilla qubits to create a double quantum dot systems with the target qubits to extract their state. In a typical double quantum dot setup, two electrons can occupy either separate dots or the same dot. When the electrons are arranged in certain spin configurations, the transition of electrons between dots can be blocked due to spin selection rules:

- If the two electrons form a triplet state (both spins aligned), Pauli exclusion prevents them from occupying the same dot, effectively blocking the tunnelling transition.
- Conversely, if the electrons are in a singlet state (opposite spins), tunnelling is allowed.

This difference in tunnelling behaviour creates a measurable difference in the charge configuration of the double dot, which can be detected by nearby charge sensors. By monitoring this charge state, the spin state of the electron pair can be inferred. Pauli Spin Blockade thus provides a method for spin-to-charge conversion in multi-spin qubit systems.

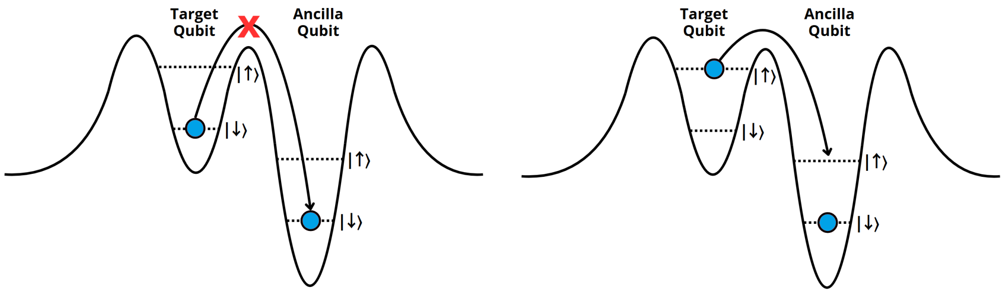


Figure 2.17: Pauli Spin Blockade readout configuration

Chapter 3

Engineering a Gate-Defined Quantum Dot

In this chapter the main structures and techniques for the engineering of gate-defined quantum dots will be illustrated. Engineering such quantum dots involves the precise design and fabrication of nanoscale gate electrodes, which, when biased appropriately, create tunable electrostatic potentials in a two-dimensional electron gas or similar semiconductor heterostructures. Understanding these engineering aspects is essential for developing reliable and scalable quantum dot devices suitable for spin qubit operation. The focus will be on gate defined quantum dots that are promising in terms of compatibility with the current state-of-the-art semiconductor processes. Thus, only structures based on SiMOS and heterostructures will be illustrated.

3.1 Material Systems

Before diving in the physical implementation of the quantum dots, it is necessary to study the material systems that will host the qubits. The choice of material system plays a crucial role in the performance and scalability of spin qubits. The intrinsic properties of the semiconductor host, such as nuclear spin environment, electron mobility and interface quality, directly impact qubit coherence times, control fidelity and device reliability. The first and most important aspect is that is to reduce to the minimum the noise given by hyperfine interactions. As described before, this type of interaction between electron spins and nuclear spins introduces a substantial quantity of noise, that will lead to decoherence much faster than in an optimum medium.
The fact is that in nature, most elements exist in a form called isotopes, atoms of the same element that have the same number of protons but differ in the number of neutrons. The different number of nucleoids do not change the material itself nor the total charge of the atom, but may introduce a variation in the nuclear spin. The best scenario for a qubit platform is a material with zero nuclear spin, so that it cannot interfere with quantum operation or contribute to decoherence speed-up. Usually, for such applications, only stable isotopes are taken into account, due to the fact that radioactive ones are not

useful for electronics, emits radioactive rays and will decay into some other element. The main stable isotopes used in standard semiconductor devices are:

Nuclide	**#protons**	**#neutrons**	**Spin**	**Natural abundance**
^{28}Si ●	14	14	0+	92.23%
^{29}Si ●	14	15	$\frac{1}{2}$+	4.68%
^{30}Si ●	14	16	0+	3.09%
^{70}Ge ●	32	38	0+	20.52%
^{72}Ge ●	32	40	0+	27.45%
^{73}Ge ●	32	41	$\frac{9}{2}$+	7.76%
^{74}Ge ●	32	42	0+	36.52%

Table 3.1: Stable isotopes of Silicon and Germanium

In table 3.1 the isotopes marked with ● represents the non null nuclear spin ones and must be avoided when developing a suitable qubit structure. The ones marked with ● are possible candidates but not utilized due to low abundance on nature or other technological aspects. Finally, the ones marked with ● are the commonly used for quantum technology. To make this compatible with standard CMOS processes, a standard wafer is used, then, taking silicon as an example, an additional layer of pure ^{28}Si is grown on top via epitaxy. This is usually called isotopically enriched silicon or isotopically pure silicon, where the concentration of other isotopic atoms is lower than $800 \div 1000$ ppm. Commercially, a purity term is used with the N-notation, where the number preceding N is the number of nines composing the percentage of the purity:

Name	**Purity**	**Impurity**
3N	99.9%	1000ppm
4N	99.99%	100ppm
5N	99.999%	10ppm

Table 3.2: Purity levels comparison

For quantum computing, a purity grade of 3N to 4N is enough to guarantee low hyperfine interactions. To grow such a isotopically enriched silicon, the precursor of silicon is needed, silane gas (SiH_4 or SiF_4) that was enriched before hand. An epitaxial growth follows to deposit a thin film of pure ^{28}Si. For instance, *ASP Isotopes Inc.* uses a technique called "Aerodynamic Separation Process" (ASP) to enrich ^{28}Si silane gas. “The ASP enrichment process uses an aerodynamic technique similar to a stationary wall centrifuge. The isotope material in raw gas form enters the stationary tube at high speed by tangential injection through finely placed and sized openings in the surface of the tube. The gas then follows a flow pattern that results in two gas vortexes occurring around the geometrical axis of the separator. The isotope material becomes separated in the radial dimension as a result of the spin speed of the isotope material reaching several hundred meters per second. An axial mass flow component in each tube feeds isotope material to the respective ends of the separator where collection of the portions of isotope material is accomplished."[2]

Several material systems are suitable for gate-controlled spin qubits, each offering particular advantages and disadvantages depending on the application. These include engineered heterostructures, where charge carriers are strongly confined along the growth direction, nanowires, which provide natural confinement in two directions and planar semiconductor platforms. [6]

Starting in 2005, the first experiments demonstrating spin qubits were reported in GaAs-AlGaAs heterostructures [36], where the two-dimensional electron gas (2DEG) in the GaAs layer is depleted by negative gate electrodes to trap individual electrons for qubit operations. The GaAs platform benefits from relatively simple fabrication and favourable electronic properties, such as a single conduction band valley and a small effective mass, which reduces lithographic constraints. However, all atoms in the lattice carry non-zero nuclear spin, making hyperfine interactions a significant source of decoherence and resulting in intrinsic inhomogeneous dephasing times of approximately $T_2^* \approx 10$ ns. Here T_2^* differs from T_2 due to Free Induction Decay (FID) that occurs when slow, inhomogeneous fluctuations are not refocused, leading to a shorter dephasing time compared to T_2. T_2^* can be extended by using spin-echo techniques, where a π-pulse (a rotation of the spin by $180°$ on the Bloch sphere) is applied, through a resonant microwave burst that generates an oscillating magnetic field at the qubit's Larmor frequency, midway through the evolution to refocus dephasing.

Currently, silicon spin qubits are among the most coherent, with gate-controlled implementations achieving (dynamically decoupled) coherence times up to 28 ms [52]. Nevertheless, challenges remain [60]. In silicon, devices must be smaller than their GaAs counterparts due to the larger effective mass of electrons and fabrication reproducibility is not yet at the same level. Valley degeneracy can lead to low-lying leakage states that may be thermally populated even at low temperatures [60]. Future research on silicon qubits must address scalability issues, as valley splitting is sensitive to unavoidable fabrication defects, inhomogeneities and step edges in nanowires, interfaces, or heterostructures [6].

3.2 Technological Evolution of Gate-Defined QDs

The development of gate-defined quantum dots began in the early 1990s with devices based on $GaAs/AlGaAs$ heterostructures. A straightforward implementation of a heterostructure-based quantum well exploits the bandgap offset between materials such as silicon, germanium or other III–V compound semiconductors. For instance, in $Si/SiGe$ heterostructures, the electrons reside in a buried quantum well [5], where electrons are naturally confined due to the conduction band offset. The heterostructure is engineered through epitaxial growth or chemical vapour deposition, allowing for precise control over material composition, layer thickness and interface quality. Gate electrodes, fabricated using the same lithographic techniques employed in MOS architectures, are then patterned on top of an insulating oxide layer to locally define quantum dots and control electron accumulation. These systems provided an exceptionally clean two-dimensional electron gas (2DEG) with high mobility and low disorder, offering an ideal platform for demonstrating electrostatic confinement at the single-electron level. Early implementations operated in depletion mode, where the 2DEG is present by default, and negatively biased gates are used to locally deplete the electron gas and define quantum dots via tunnel barriers. This approach enabled the first clear demonstrations of Coulomb blockade, charge quantization, and later spin manipulation in semiconductor nanostructures, establishing the foundational techniques that would influence all subsequent quantum-dot technologies. Nowadays, heterostructure-based devices operate in accumulation mode. In this regime, quantum dots are formed by positively biasing gate electrodes to locally accumulate electrons from the 2DEG. This approach offers several advantages, including a cleaner electrostatic environment and more flexible quantum dot design, due to the absence of parasitic conduction paths and fixed charge at the oxide-semiconductor interface. In figure 3.1 a heterostructure-based accumulation mode device representation in shown, where the plunger gates models the potential landscape to define a potential minima inside the 2DEG layer, forming a quantum dot. It is common to see implementations with a scaled-up version of these gates used to define a 2DEG reservoir, often called accumulation gates.

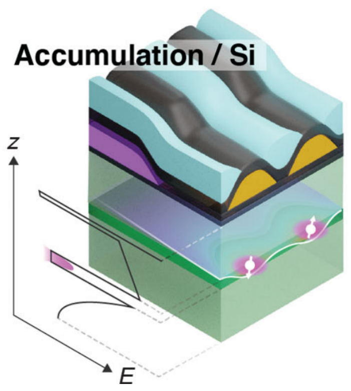


Figure 3.1: Accumulation heterostructure

Reprinted figure with permission from Guido Burkard et al., "Semiconductor spin qubits", Rev. Mod. Phys. 95, 025003 (2023). Copyright 2023 by the American Physical Society. `http://dx.doi.org/10.1103/RevModPhys.95.025003`

A more recent reinterpretation of heterostructure-based architectures is the so-called SLEDGE device. Much of the progress on silicon qubits to date has relied on gate structures fabricated using lift-off metallization techniques [53, 59], which suffers from poor wafer-level process control and has long been abandoned in mainstream silicon integrated circuit foundries, where it has not been used for several decades [22]. To overcome this problem, a research team at HRL Laboratories, led by Matthew G. Borselli, developed a new structure called SLEDGE (Single-Layer Etch-Defined Gate Electrode). This innovative approach was first detailed in a 2021 publication titled *"A flexible design platform for Si/SiGe exchange-only qubits with low disorder"* [20] published in Nano Letters. Their work introduced a flexible design platform for exchange-only qubits based on $Si/SiGe$ heterostructures, characterized by gate electrodes defined through a single-layer etching process. This design reduces electrostatic disorder compared to traditional devices. The heterostructure consists of a tensile-strained silicon quantum well epitaxially grown on a strain-relaxed $Si_{1-x}Ge_x$ $(x = 0.25 \div 0.35)$ buffer, followed by a $SiGe$ capping layer of the same stoichiometry as the buffer [20]. In traditional silicon qubit devices, gate electrodes are often formed by lift-off metallization, a process in which metal films are deposited over a patterned resist and then lifted off to leave behind the desired gate geometry. While lift-off is simple to implement at the laboratory scale, it inherently suffers from limited control over feature uniformity, edge definition and metal adhesion across an entire wafer. These shortcomings translate into variability in gate dimensions, irregular side-wall profiles and non-uniform film thickness, all of which degrade qubit performance and yield. The SLEDGE architecture directly addresses these issues by replacing lift-off gates with a single-layer, etch-defined gate electrode process. Firstly, a blanket metal film is deposited over the oxide and then patterned by a highly controllable dry-etch step, rather than relying on resist undercuts and solvent lift-off. This etch-first approach ensures uniform gate widths, vertical sidewalls and excellent alignment accuracy from die to die and wafer to wafer. The entire fabrication process is explained in detail in the original paper [20].

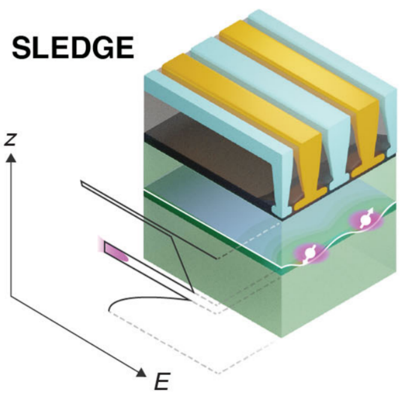


Figure 3.2: SLEDGE architecture

Reprinted figure with permission from Guido Burkard et al., "Semiconductor spin qubits", Rev. Mod. Phys. 95, 025003 (2023). Copyright 2023 by the American Physical Society. `http://dx.doi.org/10.1103/RevModPhys.95.025003`

The transition from heterostructure-based quantum dots to silicon-based devices began in the early 2000s and accelerated during the following decade. While III-V compound semiconductors devices offered high electron mobility and mature fabrication techniques, their electron spins suffered from rapid decoherence due to the presence of nuclear spins in the host material. Isotopically purified ^{28}Si provides a nuclear-spin-free environment, dramatically increasing spin coherence times up to several microseconds (free induction decay time $T_2^* \approx 120\mu s$)[52]. Moreover, silicon-based quantum dots are compatible with standard CMOS technology, enabling more straightforward scaling toward larger qubit arrays. This combination of enhanced coherence and technological compatibility has made silicon the preferred platform for spin qubits in recent years. Silicon Metal-Oxide-Semiconductor refers to a class of semiconductor devices built using a layered structure consisting of a silicon substrate, a thin insulating layer of silicon dioxide, or other dielectric materials with silicon dioxide as the interface and a conductive gate, historically aluminium or polysilicon, now replaced with metal gates. This architecture forms the basis of the MOSFET, the fundamental building block of modern digital electronics. The Si-MOS platform became dominant due to the excellent native oxide properties of silicon, enabling mass production of transistors with high yield and reliability. It laid the foundation for CMOS technology, which now underpins virtually all microprocessors, memory chips and digital logic circuits. In the context of quantum computing, Si-MOS technology offers a familiar and mature fabrication ecosystem, making it attractive for developing scalable spin qubits. Among the several Si-MOS structures, the planar one represents the most straightforward and historically established for realizing silicon-based qubits. The metal gates deposited on the surface serve different roles, typically classified as barrier gates and plunger gates, used to define quantum dots through electrostatic confinement in the x and y axis, while the interface between Si/SiO_2 provide confinement along the z axis. The planar geometry allows for relatively simple fabrication processes, as it relies on 2D lithographic patterning and deposition techniques widely used in CMOS technology. This makes the planar qubit a favourable candidate for early-stage research and prototyping. Moreover, its compatibility with standard silicon wafers facilitates integration and scalability.

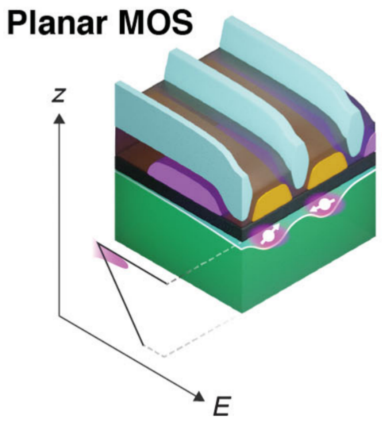


Figure 3.3: Planar structure

Reprinted figure with permission from Guido Burkard et al., "Semiconductor spin qubits", Rev. Mod. Phys. 95, 025003 (2023). Copyright 2023 by the American Physical Society. `http://dx.doi.org/10.1103/RevModPhys.95.025003`

The migration from planar to three-dimensional architectures marks a significant advancement in the development of semiconductor-based qubits. This transition brings several key advantages in terms of quantum confinement, gate control and integration density. In three-dimensional structures, the vertical confinement of the charge carriers remains governed by the Si/SiO_2 interface, as in planar devices. However, the electrostatic confinement is substantially improved due to the introduction of gate electrodes that wrap around the semiconductor channel on multiple sides. In the FinFET geometry, for instance, the gate surrounds the fin-shaped channel on three sides, top and lateral, providing superior electrostatic control over the quantum dot. This enhanced control allows for tighter confinement of single electrons and increased tunability of the dot's potential landscape. Moreover, the use of Silicon-On-Insulator wafers in conjunction with three-dimensional structures offers additional benefits. SOI technology incorporates a buried oxide layer beneath a thin silicon device layer (often called silicon overlay), electrically isolating the active region from the bulk substrate. This reduces parasitic capacitance, suppresses charge noise from the substrate and confine electrons in the thin semiconductor layer, all of which contribute to improved coherence times and better qubit fidelity. This thin silicon layer in SOI wafers so facilitates precise vertical confinement and allows for more reproducible quantum dot formation. The similarity of FinFETs to modern CMOS transistors opens the door to hybrid quantum-classical systems and monolithic integration with control electronics, potentially on the same chip. Despite the increased fabrication complexity compared to planar structures, three-dimensional qubit architectures are increasingly viewed as a promising pathway toward scalable quantum processors. Experimental demonstrations of single and multiple qubit operations in FinFET-based systems have already shown high levels of control and coherence, validating their potential for next-generation quantum technologies. One of the key challenges in this type of architecture is the integration of isotopically enriched semiconductors (such as ^{28}Si) into the device layer of SOI wafers, which is essential for reducing decoherence caused by nuclear spin noise. More on this topic is reviewed in Section 7.1.

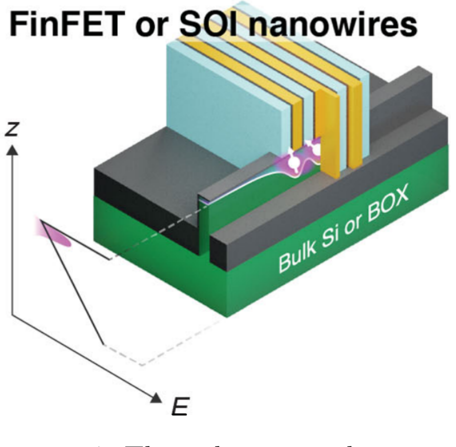


Figure 3.4: Three-dimensional structure

Reprinted figure with permission from Guido Burkard et al., "Semiconductor spin qubits", Rev. Mod. Phys. 95, 025003 (2023). Copyright 2023 by the American Physical Society. http://dx.doi.org/10.1103/RevModPhys.95.025003

3.3 State Of The Art Quantum Devices

Currently, the state-of-the-art implementation of spin qubits is predominantly based on planar silicon MOS devices operating in accumulation mode. In this configuration, electrons are accumulated under the gate, and the quantum dots are defined electrostatically. Qubits are typically encoded using the Loss–DiVincenzo single-spin scheme, although in some experiments singlet-triplet encoding is also employed. Plunger and tunnel gates can be patterned accordingly to standard architectures found in literature, while the depletion barriers, used to define the quantum dot inside an uniform 2DEG layer, can be replaced by the bandgap offset offered by the Si/SiO_2 interface, realized with Shallow Trench Isolation process.
For qubit readout, electron spins are manipulated via electron spin resonance (ESR). A static magnetic field B_0 of a few tesla ($\sim 1 \div 2$) produces a Zeeman splitting that defines the spin quantization axis. This field is typically generated by an off-chip component, such as a wire coil or an integrated coil. A dynamic magnetic field B_1 of a few millitesla, orthogonal to B_0, is used to drive coherent spin rotations. This field is generated by the application of a microwave signal to a in situ antenna, called ESR line. This can be patterned in the same way used for the other metallic gates. The readout circuitry can be either a Single Electron Transistor (SET) or a Quantum Point Contact (QPC). SOTA devices uses a SET manufactured next to the quantum dot so that they are capacitively coupled. It consists of a small conducting island connected to source and drain electrodes via tunnel barriers. They allow electrons to move one at a time, while the gate voltage controls the electrostatic potential of the island. By tuning the gate, the energy levels of the island can be aligned with the source and drain, enabling or suppressing electron tunnelling. This gives rise to Coulomb blockade conditions, where the current through the device shows discrete peaks corresponding to the addition of single electrons, making SETs highly sensitive charge sensors. When the SET is capacitively coupled to a nearby qubit, its spin state modifies the electrostatic environment of the SET, thereby changing its current. This principle allows spin-to-charge conversion and single-shot qubit readout, where the spin state of an electron in a quantum dot is detected via the change in SET current. The readout mechanism and associated circuitry are beyond the scope of this work and are therefore not considered in either the quantum or fabrication process simulations.
While single quantum dots are easier to define, simulate and manufacture, double quantum dots are studied more extensively due to several advantages for qubit control and readout. In DQDs, qubits can be encoded using singlet–triplet states or specific charge configurations, allowing spin-to-charge conversion and readout via a nearby single electron transistor. DQDs also enable precise control of the exchange interaction between electrons, which is essential for implementing two-qubit gates such as CNOT. Moreover, DQDs exhibit richer phenomena, including Pauli spin blockade and honeycomb-like charge stability diagrams, which offer detailed insight into tunnelling, capacitive coupling, and spin dynamics. In contrast, single quantum dots are limited to isolated spins, with more challenging readout and less versatile qubit operations. Due to these reasons, in this work the study of quantum dots was extended to DQDs for both the quantum and fabrication process simulations.

Part II

Implementation and Evaluation

Chapter 4

Thesis Outline

4.1 Development of a Robust Workflow for Custom Simulations

The goal of this work is to study the behaviour of custom-defined semiconductor devices as a candidate host platform for spin qubits, exploiting standard state-of-the-art fabrication processes used for CMOS and VLSI applications. In many courses in my master's degree program it was explained how to electrically characterize a device, exploiting the resulting geometry from fabrication process simulations inside Sentaurus Process. To study its behaviour in terms of quantum confinement, it is necessary to use external tools, as Sentaurus Device cannot handle this specific type of analysis. To do so, a tool from Nanoacademic Technologies named QTCAD was used. It is essentially a python-based API that provides a way to simulate quantum behaviour via Poisson, Schrödinger, Many-body, Junction and many other solvers. To simulate a device, an open-source CAD software called Gmsh is used, as addressed in the official documentation of QTCAD. It enables the definition of the geometry via its proprietary language that allows the definition of points, curves and volumes. Surfaces and volumes of the resulting geometry are then properly labelled to reflect boundary conditions and physical materials to be simulated inside QTCAD. If the workflow wants to be adapted in order to define parametric and complex geometries, it is necessary to define the geometry outside Gmsh, as these features are not natively supported and defining such geometries using its proprietary code would be tedious and time-consuming. The device geometries were defined in a powerful CAD software from Autodesk, called Fusion. It allows parametric sketching and extrusion, generating a three-dimensional model that can be resized or reshaped using numerical parameters. Since the use of external CAD files inside Gmsh is possible but not natively supported, a workaround was exploited to instruct the graphics kernel OpenCascade on how to handle an external geometry, making it compatible with the assignment of physical groups to surfaces and volumes, which is essential to obtain realistic results inside QTCAD. Once a strong and reliable way was found to correctly mesh a device geometry, many simulation scripts were created in order to simulate the architecture and give insight about the creation and occupancy of quantum dots.

4.2 Study of Planar Devices Quantum Behaviour

Analysing how a custom defined device act as a platform for a quantum dot is crucial. The study of quantum confinement is performed in QTCAD by simulating devices that models their potential landscape with barrier (or tunnel) and plunger gates. This is computed with the Poisson solver, while the Schrödinger one enables the visualization of the device wavefunction and probability density function along certain directions. The obtained data is then post-processed in order to retrieve important plots, supported by input or other retrieved parameters, that describes the potential landscape and wavefunctions along the main planes. The results can be exported in VTU format to be visualized as a colormap applied to the simulated mesh inside a software called ParaView.

Quantum confinement alone is not sufficient, as it only reveals how the quantum dot behaves under certain static conditions. A transport analysis is then performed, where the source contact act as carrier reservoir and the dynamic application of different voltages provides insight into the dot's carrier occupancies for both single and multidot configurations. This ensures that the device behaves as expected and can be considered a valid candidate to host spin qubits.

4.2.1 Moving to Three-Dimensional Architectures

Once the simulation scripts demonstrate reliability for both static and dynamic scenarios, the investigation can be extended to three-dimensional architectures, such as FinFETs and Gate-All-Around FETs (here the FET suffix is intended as a reference to the original architecture).
As for MOS devices, the three-dimensionality provides a way to apply an electrostatic control by the gates that is spread over a larger surface with respect to planar devices, thus enhancing quantum confinement and shielding from external sources of noise and decoherence due to a shared substrate.

4.3 Device Fabrication Process Simulation

After a given device is simulated in QTCAD and its quantum confinement guaranteed, the physical dimensions and architecture are transferred into Synopsys Sentaurus Process to simulate its fabrication process. Since the fabrication simulation of three-dimensional architectures is complex and the readout of a single quantum dot device cannot be carried out without a quantum point contact or by exploiting Pauli exclusion principle, a double quantum dot device was chosen for process simulation inside SProcess, as stated in Section 3.3. This phase is essential to provide realistic insight about the feasibility of manufacturing, especially for a low-volume research laboratory as PiQuET.

Chapter 5

Preliminary Setup

The primary objective is to analyze the device in terms of quantum confinement and technological feasibility. To do so, it is necessary to define a model of the desired device. This translates into the modelling of a geometrical representation of the device. The resulting mesh is then used in the quantum simulation environment. The following sections present the main techniques used to achieve these goals.

5.1 Workflow Outline

The first aspect to be addressed is the selection of the target device type. This work aims at adapting current state-of-the-art VLSI technological processes for the fabrication of a MOS-like structure capable of hosting spin qubits. It was therefore decided to adapt a planar MOS structure by adding extra gates to function as barrier and plunger gates, in order to ensure carrier confinement within the silicon channel. As stated in section 3.3, depletion barriers were substituted by Shallow Trench Insulation. The first step is to define the device geometry in a CAD software and then convert it into a suitable mesh format. This mesh is subsequently used in the QTCAD simulation environment to extract information about quantum confinement. Once confinement is confirmed by the simulation results, it is possible moving to Synopsys Sentaurus to perform a preliminary analysis of the fabrication process and assess the feasibility of the device at the nanoscale. Additional constraints were added due to the capabilities of the fab that Politecnico di Torino relies on, a research laboratory in Turin called PiQuET, Piemonte Quantum Enabling Technology. More on this topic is covered in the dedicated section 7.2.
A lot of geometries were made to be simulated in QTCAD, with different dimensions and technological features (planar, Fin, Gate-All-Around). To illustrate the modelling and meshing steps for the next sections, a simple planar SQD SOI MOS-like device will be used. It consist of an adaptation of a typical SOI planar MOSFET with the addition of two additional gates, so that the configuration for quantum confinement can be guaranteed by the plunger gate, that bends the energy band to create a local minima and the two barrier gates, that, in a single-dot configuration, act as barriers toward the lead contacts. In a multidot configuration they also serve to tune the interdot coupling.

5.2 Geometry Definition with Autodesk Fusion

Fusion, sometimes referred as Fusion360, is a versatile CAD software from Autodesk widely used for designing products and simulating mechanical structures. One of the key advantages of the software is its capability to define parametric sketches, allowing precise control over dimensions and relationships between geometric features. This makes it particularly suitable for designing tree-dimensional geometries with single features that can be resized automatically simply by changing its parameters. In this work, it is employed under a free personal non-commercial license.
Fusion, unfortunately, has a steep learning curve and can take several years to master fully. However, for the purpose of this work, it is only necessary to learn how to create sketches and how to extrude and combine three-dimensional volumes. It is important to keep in mind that the choice of this software is completely personal and any other CAD software can be used to define the device geometry. The software was selected due to the candidate's extensive experience with it and the considerable time savings offered by parametric design. If it is needed to simulate the exact same device but with a different feature size, the parametric drawing allows to get the new design by modifying the parameters, without the need to model the device again. Furthermore, the parameters are saved in a `.csv` file that can be exported along the geometry to automatically obtain significant physical dimensions inside the quantum simulation environment.

5.2.1 Parameters Definition

A 2D drawing placed in a specific plane, inside Fusion, its called a sketch. From the resulting surfaces it is possible to extrude volumes, so it is the first thing to do. Before start drawing inside the software it is necessary to sketch by hand a rough shape of the device. This is useful to note all the desired physical dimensions to be parametrized.

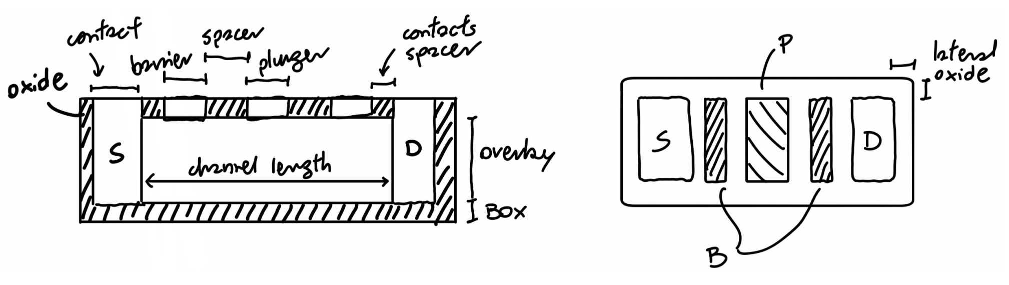
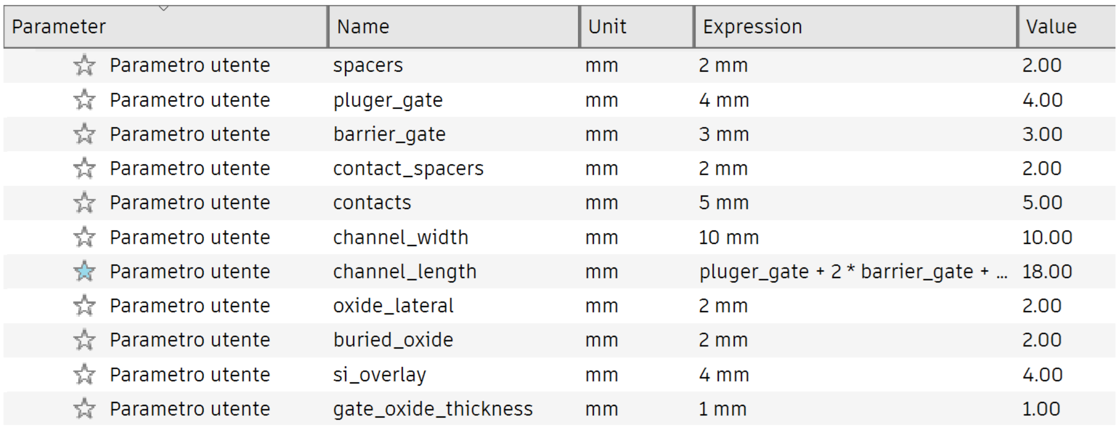


Figure 5.1: Device pen&paper model

It is important to note that QTCAD does not support metals, the contacts are so directly placed on oxide or semiconductor surfaces and then the metal workfunction can be defined in order to simulate different metal contacts. It is so necessary to not model the metal gates but to define the oxide region where an hypothetical metal gate will sit on. If no separation of the oxide volume is done then the resulting geometry will lack a surface where to place the contact, as explained in Section 5.3.2.

At the end, the desired parameters for the device are:



Parameter	Name	Unit	Expression	Value
☆ Parametro utente	spacers	mm	2 mm	2.00
☆ Parametro utente	pluger_gate	mm	4 mm	4.00
☆ Parametro utente	barrier_gate	mm	3 mm	3.00
☆ Parametro utente	contact_spacers	mm	2 mm	2.00
☆ Parametro utente	contacts	mm	5 mm	5.00
☆ Parametro utente	channel_width	mm	10 mm	10.00
★ Parametro utente	channel_length	mm	pluger_gate + 2 * barrier_gate + ...	18.00
☆ Parametro utente	oxide_lateral	mm	2 mm	2.00
☆ Parametro utente	buried_oxide	mm	2 mm	2.00
☆ Parametro utente	si_overlay	mm	4 mm	4.00
☆ Parametro utente	gate_oxide_thickness	mm	1 mm	1.00

Figure 5.2: Device parameters

Notice that the parameter `channel_length` has as expression composed of other parameters, in particular:

$$pluger\_gate + 2 * barrier\_gate + 2 * spacers + 2 * contact\_spacers$$

Thus, the channel length is not a fixed value but the sum of the dimensions of the gates and the spaces between them. This approach allows the channel length to scale consistently with the gate contacts.
The last column shows the value, which is the result of the expression. This value is dimensionless because the software interprets it according to the project's standard measurement unit, in this case millimeters. The choice of millimeters was mandatory: using an SI unit such as meters would result in an exported model scaled by a factor of 10^3, since millimeters are the default metric unit in Fusion (or inches if the imperial scale is selected). Modelling in millimeters ensures that the exported geometry retains the intended dimensions without any scaling. In this context, it is possible to work in millimeters while treating them as nanometers, as will be specified later in the simulation environment.

After identifying all the desired parameters it is possible to add them to Fusion workspace by selecting, in the Solid tab, `Modify > Change Parameters` and then adding one by using the plus icon.

5.2.2 Parametric Sketching

It is possible to select one plane and start a sketch. To do so in Fusion, use the Create Sketch feature and selecting the YoZ plane it is possible to define the lateral profile of the device, then extruding it in the x axis.

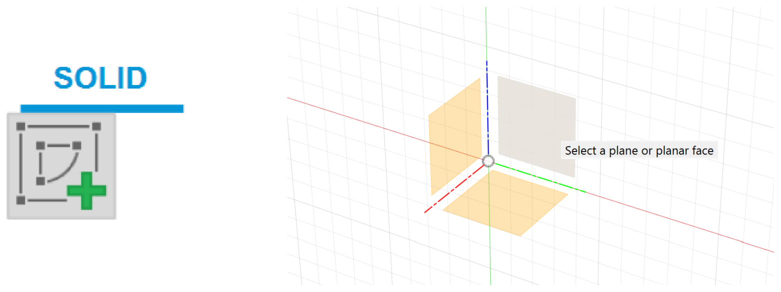


Figure 5.3: Sketch Selection

For this example, it is just needed to define the channel profile, the lateral oxide and the top gates oxide in a single sketch, as shown in figure 5.4.

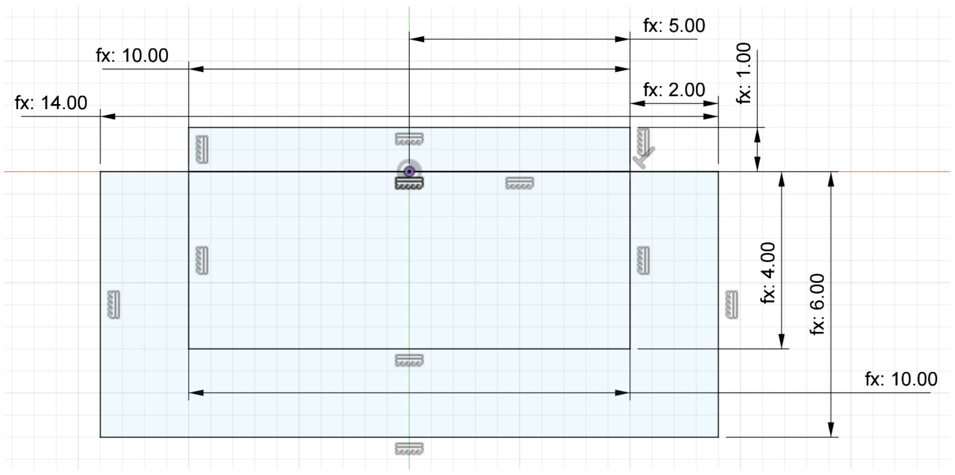


Figure 5.4: Device YoZ Sketch

Each dimension is labelled as “fx: xx.00”, the “fx” appendix shows that the final number is not fixed but is the result of a n expression. For instance, the “fx: 10.00” that appears in the upper-left corner is the result of the parameter `channel_width`, while the “fx: 14.00” below the result of the expression “`2 * oxide_lateral + channel_width`”. To ensure coherence across all workflow steps, the origin of the sketch, and so of the geometry, was set at the expected dot location, in case of a fully symmetrical device, while the channel protrude along the x direction. For this device, the geometry will have symmetry along the x- and y-axis, but not on the z axis. In this case, the origin was set at the center of the channel at the interface between the latter and the gate oxide. For this simple example, this sketch is all that is needed to get the full device and it is ready to be extruded.

5.2.3 Parametric Extrusion

Once the sketch is complete, the extrusion feature can be used to generate a volume from a surface. The extrusion steps must be performed parametrically to preserve the original intention of creating a resizeable geometry. In figure 5.5 it is possible to see how each extrusion takes a dimension parameter. This can be set to the name of a parameter, or a generic expression, to be extruded as intended.

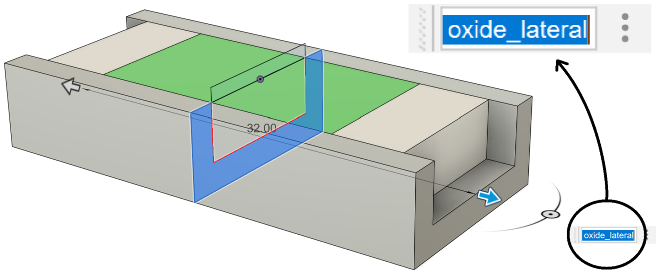


Figure 5.5: Parametric extrusion along the x axis

In figure 5.6 it is possible to observe the 6 main extrusion steps to get the full geometry:

1. Extrusion of the channel
2. Addition of lead contact areas to the side of the channel
3. Addition of lateral oxide
4. Addition of gates (blue and red) and spacers (momentarily gray)
5. Raising of lateral oxide and spacers merge
6. Addition of lead contacts lateral oxide

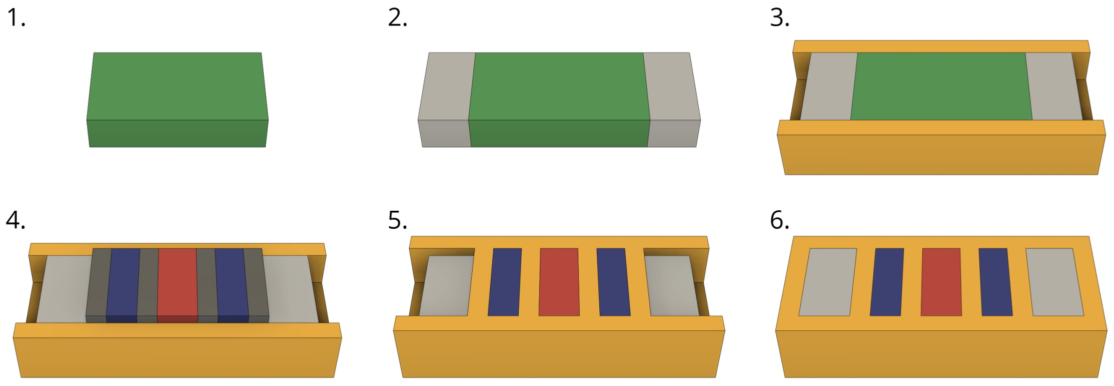


Figure 5.6: Extrusion steps

In figure 5.6 the yellow area is the oxide, the green one the silicon channel, the red one the plunger gate, the blue ones the barrier gates, the light gray ones the contacts and the dark grey ones are the oxide areas that separate the gates and will be merge to the total oxide volume, becoming yellow. This includes the buried oxide, the shallow trench isolation and the gates and contacts spacers. The colors were manually selected and assigned within the CAD software.
Once the device is completely extruded, a section analysis can be done to check the correctness of the volumes.

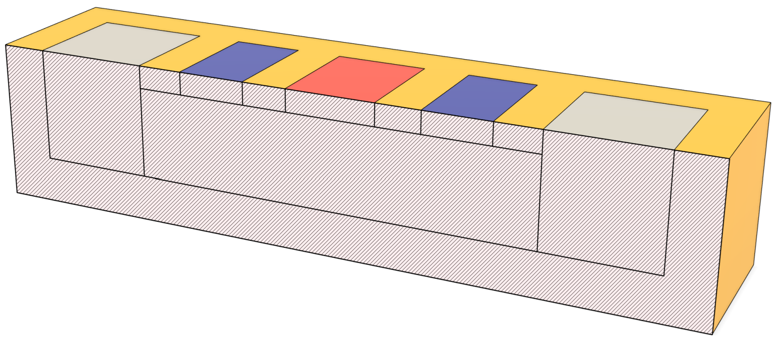


Figure 5.7: Section analysis

5.2.4 Design Export

Now the geometry is ready to be exported. For reasons explained in the next section, it is necessary to export the CAD file in two different extensions. Go to the upper-left file-shaped icon and select Export, choose a suitable name and then export as a `.STEP` file. Then repeat the process and export in `.IGES` file. The two files must have the same name but different extensions. Then go to the parameters window and export them by clicking the file-shaped icon with the arrow pointing out of it. Select the `.CSV` extension and give it the same name used for the geometry file.

The export procedure generates three files:

- `device_name.step` - STEP geometry file
- `device_name.iges` - IGES geometry file
- `device_name.csv` - Exported parameters

Now the geometry files are ready to be processed into GMSH.

5.3 Geometry Meshing with Gmsh

Gmsh is an open-source 3D finite element mesh generator with a built-in CAD engine and post-processor. It is widely used for creating complex geometries and generating high-quality meshes for numerical simulations, particularly in finite element analysis. In this work, it is used to generate a suitable mesh for simulation in QTCAD. Three-dimensional meshes are generated starting from a geometry, a representation of an object using points, curves, surfaces and volumes. Gmsh allows assigning physical properties to each element, enabling accurate finite-element simulation. In practice, volumes are labelled to specify the material they consist of, while surfaces are labelled to define boundary conditions, such as gate or lead contacts. The type of contact (Ohmic, Schottky, etc.) can be defined later in QTCAD simulations (as shown in section 6). The main steps for correctly meshing a geometry coming from an external CAD are now explained in details.

5.3.1 Conformal Geometry Loading

The workspace of Gmsh is connected to a `.geo` file. It is necessary to create a new geometry file and ensure the workspace is related to the latter. From this point forward, the code will be presented and explained line by line, offering a step-by-step guide for its construction. The first step is to load the preferred kernel, in this case OpenCascade, a powerful open-source geometry engine that enables advanced modelling operations such as Boolean operations, extrusion and complex surface handling. It is now important to define the tolerance of the boolean operator. This is crucial for the next steps. Now, the mesh can be loaded by the command `Merge`.

```
SetFactory("OpenCASCADE");
Geometry.ToleranceBoolean = 1e-3;
Merge "geometry_file.step";
```

The following command is crucial for the correct processing of the geometry. The command `Coherence` calls the integrated boolean operator inside OpenCascade and performs a series of operations to make the geometry **conformal**.

```
Coherence;
```

The issue is that the import of a geometry defined in a `.step` or `.iges` file will load into Gmsh a model that includes overlapping elements. This often includes lines, curves and surfaces. If overlapping elements are exported to make quantum simulations, boundary conditions and physical volumes properties may be set to both a surface/volume belonging to a specific element of the geometry and another one that should not. This is noticeable by enabling the curve or surface labels view in `Tools > Options` and then `Geometry > Visibility > Curve Labels` or `Surface Labels`. It is possible to observe elements overlap in figure 5.8, by zooming into the geometry and search for overlapping labels. Unfortunately, this is the only way to notice an element overlap.

Line 926 (OCC)
Plane 52 (OCC)

Figure 5.8: Left: Lines/curves overlap. Right: Surfaces overlap.

In order to completely overcome this issue, as of today, the only way is to play with the parameter `Geometry.ToleranceBoolean`, which start from a default value of $1e-6$ and increase it until all of the overlapping elements are merged by the boolean operations did by the kernel via the `Coherence` command. Visual inspection is the only way to see if the command run successfully without degenerating the mesh shape. By steps of decades, increase the parameter if there are still overlapping elements. If geometry deformation starts to occur it is necessary to lower and fine tune the tolerance.

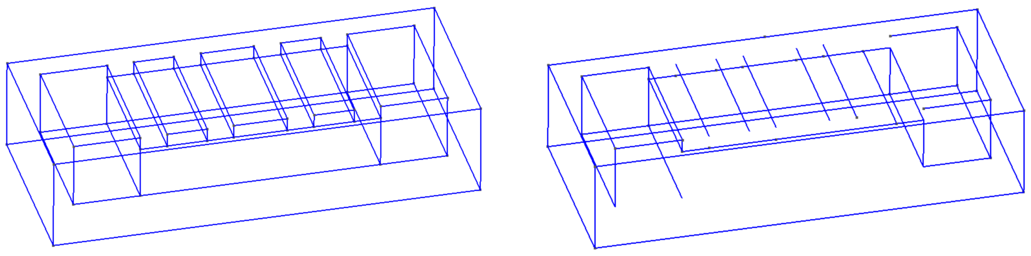
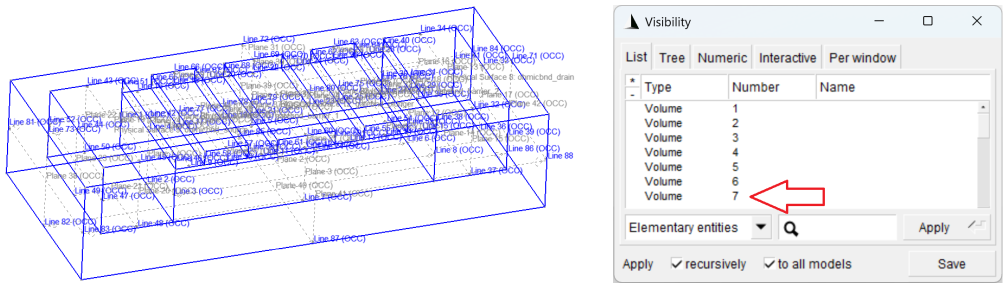


Figure 5.9: Left: Correct geometry ($tol = 0.01$). Right: Degenerated geometry ($tol = 1$).

The term conformal refers to a geometry which does not have overlapping elements and can be correctly processed for quantum simulations. The tolerance parameter must be set correctly and may vary from geometry to geometry. The presence of curves (not straight lines) seems to play a role in the correct function of the `Coherence` command. The Gmsh console log should not raise any errors and preferably nor any warnings. Some geometries may work even with the raise of the warnings *BOPAlgo_AlertSelfInterferingShape*, *BOPAlgo_AlertTooSmallEdge*, *BOPAlgo_AlertBadPositioning* and *BOPAlgo_AlertSelf*. The final check must be done by counting the number of volumes before and after the use of the `Coherence` command. It is possible to edit the file and reload it in Gmsh by using the left lateral menu `Modules > Geometry > Reload script`. Open the visibility windows, `Tools > Visibility > List` and then, in the bottom left menu, order by elementary entities. By default the order starts with the volumes and with the associated number. Here it is possible to see the number of volumes and check that by modifying the tolerance parameter this number does not change. A different number of volumes almost always translates to an incorrect conformal conversion.


Visibility

List | Tree | Numeric | Interactive | Per window

Type	Number	Name
Volume	1	
Volume	2	
Volume	3	
Volume	4	
Volume	5	
Volume	6	
Volume	7	

Elementary entities | Apply

Apply ✓ recursively ✓ to all models | Save

Figure 5.10: Left: Conformal geometry. Right: Volumes count check.

Once again, it is possible to check if the scaling and axes origin are set correctly by loading the geometry. Open `Tools > Options > General > Axes > Axes mode` and select `Full grid`. In figure 5.11 is is possible to observe that the zero (or very small values) represents the origin of the relative planes and that the axes origin correctly meet the origin point chosen in the modelling step.

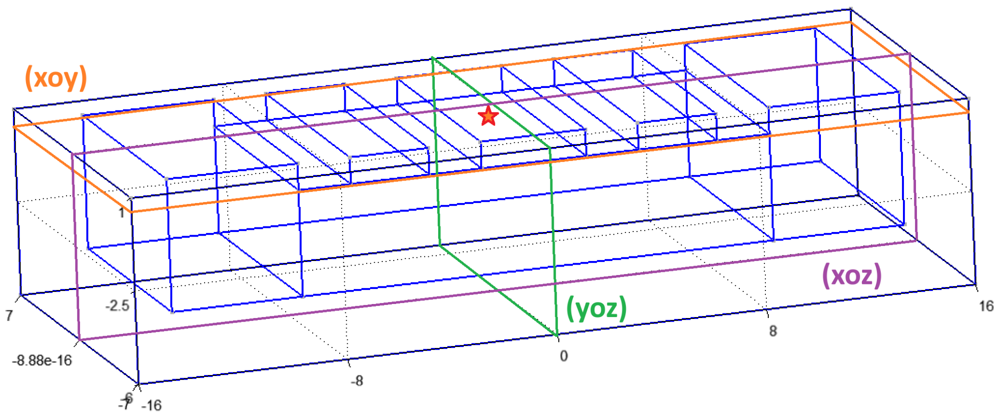


Figure 5.11: Geometry with full grid axes shown. The origin is marked with a red star and each plane is highlighted with a different colour.

The grid encapsulating the geometry also shows the coordinates of the boundaries. Here it is possible to check that the dimensions are the same with respect to the model in the CAD software where the geometry was defined. If the scale is off (e.g. $16e3$ instead of 16) it is probable that the scale set in the CAD software is different with the one set with Gmsh. Here is necessary to scale the geometry inside Gmsh or to go back to the CAD and specify a different unit for the workspace and then export the model again. A correct geometry must have coordinate numbers without any scale applied (e.g. 16, not $16e-9$) so it can be correctly loaded in QTCAD and can be scaled to the desired unit (nm, μm, etc.) there. If everything is correctly set, it is now possible to move to the assignment to physical groups.

STEP vs IGES: Guidelines for Choosing the Right Format

By applying the `Coherence` command within the OpenCascade kernel, Gmsh attempts to remove duplicate elements and generate a conformal geometry. This procedure often appears to work more reliably with IGES files compared to STEP and the reason lies in how the two formats represent geometry and topology, as well as how OpenCascade interprets them.

- STEP is designed as a modern standard to encode full B-Rep (Boundary Representation) solids, a way of describing an object by storing its boundaries, such as faces, edges and vertices. Each face belongs to a closed shell and is unambiguously linked to its adjacent edges and vertices. When two solids are adjacent, their contact faces are defined twice, each as part of a different shell. During coherence, OpenCascade must recognize that these faces are not only geometrically coincident but also topologically redundant. This recognition is sensitive to numerical tolerances and small discrepancies in the parametrization of the surfaces. As a result, coincident faces are not always merged successfully.
- IGES, on the other hand, is an older and less structured format. It typically encodes collections of trimmed NURBS (Non-Uniform Rational B-Splines) surfaces rather than topologically closed solids. NURBS represents features like flexible curves or surfaces controlled by a set of points, where each point influences the shape with a certain weight. When imported into Gmsh, the kernel reconstructs the topology directly from the raw geometric definitions. This reconstruction phase often allows coincident surfaces to be detected and unified more easily. The lack of rigid topological constraints in IGES seems to makes it simpler for the kernel to identify duplicates during the coherence operation, even though the format itself is less robust for representing solid models.

Another important factor is tolerance management. STEP preserves topological definitions very precisely, so if the contact faces differ by only a small offset or parametrization, they may still be treated as distinct entities. IGES, on the other hand, it is imported deriving the topology from scratch, which can reduce inconsistencies and favour merging.

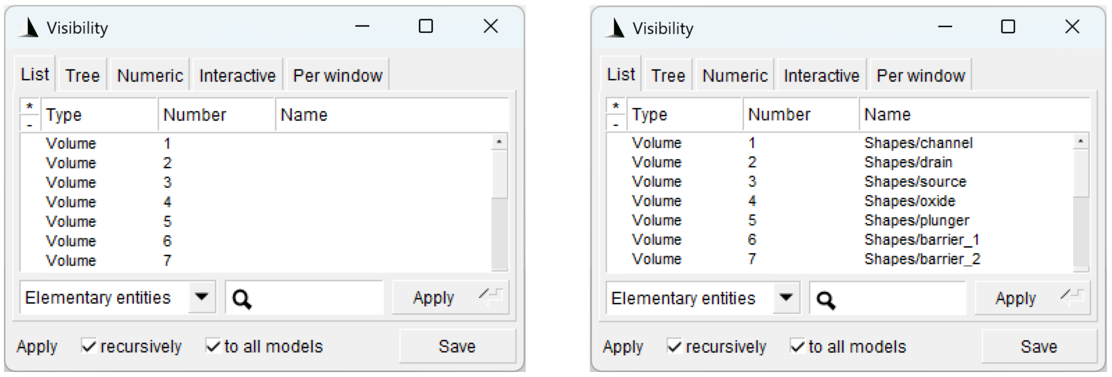
While careful tuning of the tolerance parameter is often necessary, persistent issues in the conformal conversion of the geometry may be resolved by switching to the alternative format. Therefore, it is advisable to test both formats in practice. The final verification should be performed during the assignment of physical groups to ensure that all intended labels are correctly recognized. Further details are provided in the following section.

5.3.2 Physical Groups Assignment

These steps aim to assign the appropriate labels to each volume and selected surfaces, ensuring their correct recognition within QTCAD. This process is essential for accurately defining material properties and boundary conditions within the simulator. Although there are several methods for setting these attributes, the most straightforward approach is outlined below, with a distinction between volumes and surfaces.

Volumes

Each volume should be labelled in order to specify the material later on. So if a mesh have n volumes, it is compulsory to label n of them. The assignment can be done manually inside the Gmsh GUI. In this case, the software highlights with a small sphere the center of each volume and it is possible to perform assignment by typing the desired name and then clicking the corresponding volume center. The issue with symmetrical geometries is that many volumes may have the same center and so they may overlap while choosing the correct volume, by selecting the corresponding center. A better way is to merge the `.iges` type, which preserves the bodies names defined inside the CAD software, so it is possible to see, in the visibility window, the volume number and the relative name.



Type	Number	Name
Volume	1	
Volume	2	
Volume	3	
Volume	4	
Volume	5	
Volume	6	
Volume	7	

Type	Number	Name
Volume	1	Shapes/channel
Volume	2	Shapes/drain
Volume	3	Shapes/source
Volume	4	Shapes/oxide
Volume	5	Shapes/plunger
Volume	6	Shapes/barrier_1
Volume	7	Shapes/barrier_2

Figure 5.12: Left: STEP file merge. Right: IGES file merge.

Even if, for the reasons explained in Section 5.3.2, it may be necessary to switch from the IGES type to STEP type, the numbering does not change. So it is possible to always start by loading the IGES file, define physical groups for the volumes and then switch to the format that guarantees a correct meshing procedure. A generic volume n remains the same in both formats because the numbering reflects the order in which the volumes were defined in the CAD software. Extra care should be taken while assigning surface physical groups, as the numbering of the surfaces may vary with file type and resulting geometry from the use of the `Coherence` command, which is critically reliant on the tolerance parameter. The last statement seems to be particularly valid in the context of complex geometries that feature curved or round surfaces (e.g. nanowires).

The command to assign a physical group to a volume is:

```
Physical Volume("label_name") = {volumes};
```

It is possible to place inside `volumes` the number (or numbers, separated by a comma) of all the volumes to be labelled with that name. An example of a single quantum dot with two barrier gates and a plunger gate physical group assignment follows:

```
Physical Volume("semi_channel")   = {1};
Physical Volume("nsemi_source")   = {3};
Physical Volume("nsemi_drain")    = {2};
Physical Volume("diel_oxide")     = {4};
Physical Volume("diel_plunger")   = {5};
Physical Volume("diel_barrier_1") = {6};
Physical Volume("diel_barrier_2") = {7};
```

Here, the gate oxides for the plunger and the two barriers were defined separately with respect to the `diel_oxide` tag (which includes buried oxide, Shallow Trench Isolation and contacts spacers volumes), allowing the assignment of different types of dielectric if needed.

The prefixes `semi_`, `nsemi_`, `psemi_` and `diel_` are used, along with the relative ones defined for the surfaces, in order to automatically generate a QTCAD-API Python code. This code can then be copied and pasted into the simulation script to streamline the setup process and avoid unnecessary manual work. To do so a Python script called `device_config.py` has been created. It can be seen in the attachments appendix as script A.1.
It can be called by the command

```
python device_config.py input_file.geo
```

and takes as input the Gmsh geometry file `.geo` and generates a text file called `device_config.txt` containing all the code to be copied into the simulation script. The use of this program is completely optional.

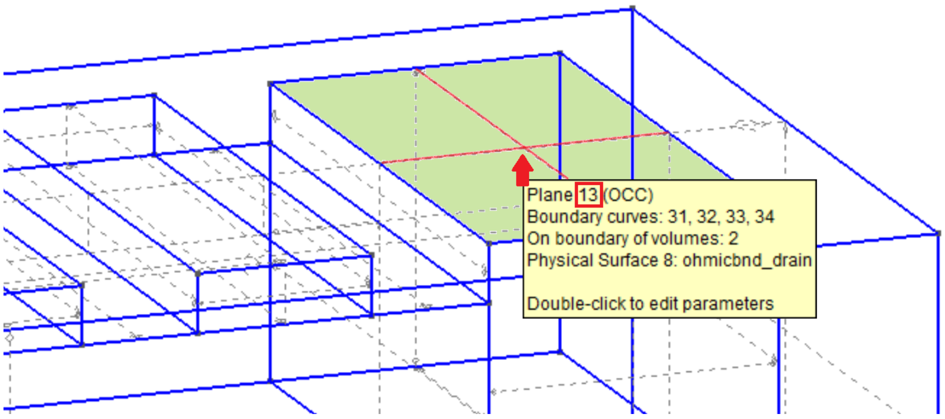
Surfaces

Unlike the assignment of volumes, not all surfaces should be labelled. Here the goal is to assign surfaces in order to represent contacts where voltages will be applied later in the simulation environment. Mainly, if working with a MOS-like structure the main elements are the source and drain contacts, the gate contacts and, if needed, a backgate contact. QTCAD does not support metals, the contacts are so directly placed on oxide or semiconductor surfaces and then the metal workfunction can be defined in order to simulate different metal contacts. QTCAD supports different types of contacts, as reviewed in Chapter 6, but in this step, to perform a correct physical assignment to surfaces, it is not necessary to know which type. It is possible to assign a surface physical group to each element that need a contact.

The command to assign a physical group to a surface is:

```
Physical Surface("label_name") = {surfaces};
```

Like for the assignment of physical groups to volumes, it is possible to define `surfaces` with the number of each surface composing the contact. In this case, the meshing of a planar device, all the contacts are plane surfaces and so only one number is defined. In complex geometries it is often necessary to specify multiple surface numbers, separated by a comma. Here, the visibility window, used for identify the volume numbers with ease, does not provide any help regarding the identification of surfaces. The correct surface number should be picked manually. To do so it is necessary to enable surface visibility, `Tools > Options > Geometry > Visibility` and check the `Surfaces` option. Dotted grey lines will appear to show the surfaces, defined by blue lines. Placing the cursor on the dotted lines of a surface shows information about the latter, including the surface number.



Plane 13 (OCC)
Boundary curves: 31, 32, 33, 34
On boundary of volumes: 2
Physical Surface 8: ohmicbnd_drain

Double-click to edit parameters

Figure 5.13: Manual selection of the surface of the drain contact. False colour: highlighted in green is the surface, in red the dotted lines. The surface number, in this case, is 13.

Therefore, by placing the cursor on the lines the surface number is shown and it can be placed in the assignment of surface labels. For the same example, a single quantum dot with two barrier gates and a plunger gate, the physical group assignment follows:

```
Physical Surface("ohmicbnd_drain")     = {13};
Physical Surface("ohmicbnd_source")    = {19};
Physical Surface("gatebnd_barrier_1") = {44};
Physical Surface("gatebnd_barrier_2") = {45};
Physical Surface("gatebnd_plunger")    = {43};
```

As for the volumes labelling, the prefixes `ohmicbnd_` and `gatebnd_` are used in the `device_config.py` script in order to speed up the generation of the QTCAD-API Python code.

Assignment Check

The final check should be carried out in order to guarantee that the physical groups assignment was successfully. This new code line is so added:

```
Mesh.ColorCarousel = 2;
```

This set the colouring mode of the mesh according to physical groups. To perform a preliminary meshing to check this condition a large element size is defined (details on section 5.3.3) in order to generate a coarse grain mesh.

```
Mesh.MeshSizeFactor = 5;
Mesh.MeshSizeMax = 1;
```

These commands allows to perform a preliminary meshing within seconds. It is now possible to reload the script and perform a 3D meshing by selecting in the left menu of the GUI `Modules > Mesh > 3D`. Then it is necessary to show the mesh faces, `Tools > Options > Mesh > Visibility` and check `3D element faces`.

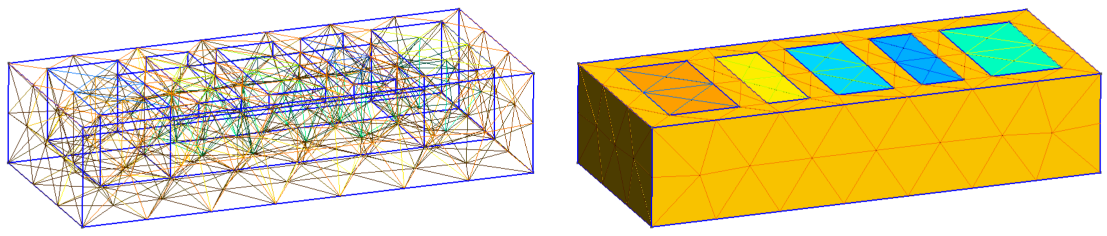


Figure 5.14: `3D element faces` - Left: Before enabling. Right: After enabling.

It is also possible to uncheck the tag `2D element edges` to better visualize the mesh. The colours are set by the default value of Gmsh, but, for a better and consistent view of the mesh, it is possible to define custom colours which reflects material, matching with the colours in the CAD software.

```
Mesh.Color.Zero =  {200, 200, 200};  // NO PHYSICAL GROUP - GRAY
Mesh.Color.One =   {150, 255, 150};  // CHANNEL - LIGHT GREEN
Mesh.Color.Two =   {  0, 255,   0};  // SOURCE - GREEN
Mesh.Color.Three = {  0, 255,   0};  // DRAIN - GREEN
Mesh.Color.Four =  {255, 190,  60};  // OXIDE (BURIED + STI) - ORANGE
Mesh.Color.Five =  {255,   0,   0};  // PLUNGER - RED
Mesh.Color.Six =   {  0,   0, 255};  // BARRIER 1 - BLUE
Mesh.Color.Seven = {  0,   0, 255};  // BARRIER 2 - BLUE
```

The colour `Zero` represents elements which are not assigned to any physical group. Even if all volumes were labelled, errors in the `Coherence` command may lead to incorrect labelling. This is noticeable by looking at the mesh and notice areas with the same colour defined for the zero group (gray in this case). All the other numbers reflects the physical groups. It is possible to check for the physical group number in the visibility windows and group by physical group (menu on the bottom left) but the most straightforward way to know the physical group number is to look at the order the volumes were labelled. The channel was the first one, so `Mesh.Color.One` represent the colour of the group called `semi_channel`.
The very simple geometry used in this example does not manifest any gray area. An example of a wrong assignment is shown in figure 5.15 with the meshing of a FinFET-like structure. The solution to this problem for this structure was to switch from IGES to STEP format.

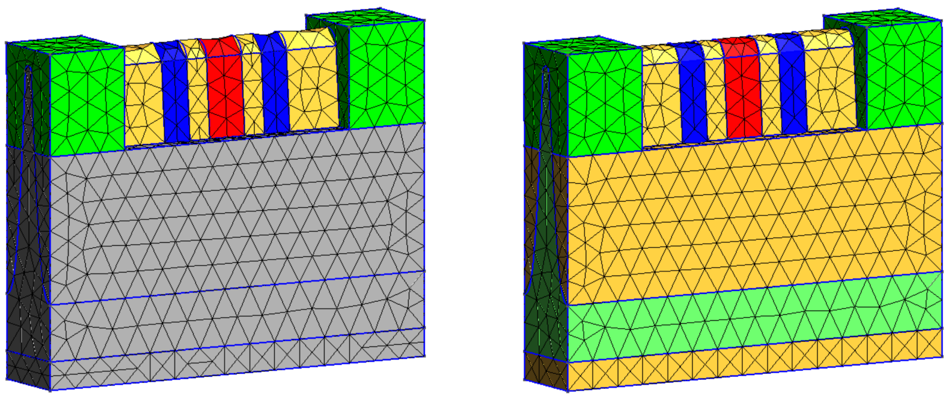


Figure 5.15: FinFET-like device with physical group colouring mode. Left: incorrect assignment (IGES format). Right: correct assignment (STEP format).

5.3.3 Meshing and Export

Meshing refers to the process of discretizing a geometric domain into a finite set of elements (such as triangles, quadrilaterals, tetrahedra, or hexahedra) suitable for computational analysis. This step is fundamental in methods like the Finite Element Method (FEM) or Finite Volume Method (FVM), where the accuracy and stability of the solution strongly depend on the quality of the mesh.
The two main parameters that dictates the resolution of the output mesh are:

```
Mesh.MeshSizeFactor = 1;
Mesh.MeshSizeMax = 0.2;
```

To determine the actual mesh size at any given point in the model, Gmsh evaluates different mesh size constraints and selects the smallest value. The resulting value is further constrained in the interval `[Mesh.MeshSizeMin, Mesh.MeshSizeMax]`. The resulting value is then finally multiplied by `Mesh.MeshSizeFactor`. By setting the mesh size factor to 1 it is possible to tune the mesh granularity by the parameter `MeshSizeMax`. In the context of solid state quantum simulations, with no scale applied to the geometrical coordinates, 0.2 represent a good starting point to obtain trustworthy results. An additional Python script called `mesh_volume.py` was created to calculate the mesh volume. This was done to calculate the minimum number of nodes the mesh must have in order to fulfil a density constraint. For instance, if a density of 100 nodes per square nanometer is set, the mesh showed in this workflow, with a volume of $3136nm^3$, must have at least 313.600 nodes.
The meshing step is computationally intensive. For this simple mesh, Gmsh took 48 seconds to allocate 311.087 nodes on an AMD Ryzen 9 8945HS processor and occupied 1,2GB of RAM with the parameter `MeshSizeMax` set to 0.2, generating a 92,7MB mesh file. Geometries with higher volumes and/or lower value of the latter parameter seems to increase the meshing time and output file size in a very fast way.
The meshing is divided in three steps, 1D, 2D and 3D. Each step quantize a degree in space and allocates nodes along lines, surfaces and volumes, accordingly. It is possible to perform each step individually or to do all three together by directly choosing the 3D meshing. To do so select in the left menu `Modules > Mesh > 3D`. It is possible to look at the progress in the console log, accessible by clicking the bar at the bottom of the screen.

Once the meshing algorithm ends it is possible to export the mesh via `File > Export`. Choose a suitable name and type the extension `.msh2`, which is one of the two mesh formats compatible with QTCAD along with `.msh4`, even if the first one (version 2) seems to work better in a variety of conditions (version 4 is slightly more compressed and sometimes its import in QTCAD raises errors). Then a pop-up window will appear, select **Version 2 ASCII** format and **leave unchecked** the two options *Save all elements* and *Save parametric coordinates*. The mesh has been correctly exported and it is ready for simulation.

Chapter 6

Quantum Simulations With QTCAD

QTCAD (Quantum-Technology Computer-Aided Design) is a simulation platform for quantum technology developed by Nanoacademic Technologies Inc. [23] It combines finite element and atomistic modelling, allowing multiscale analysis of quantum devices such as spin and superconducting qubits. Its modules cover electrostatics, capacitance extraction, Maxwell eigenmode analysis, valley splitting, g-tensor evaluation, many-body Coulomb interactions and transport through master equation or nonequilibrium Green's function approaches. QTCAD also supports cryogenic simulations and the recently added realistic atomistic models that include disorder and strain. Practically, it consists of a Python API through which complex simulation environments can be developed. QTCAD is a demanding simulation platform and the hardware requirements depend on the complexity of the simulations. For basic tasks with relatively small meshes, a standard modern processor, around 8 GB of RAM and a solid-state drive provide sufficient performance. More complex simulations, involving large meshes and multiple quantum devices, benefit from a high-performance processor and several GB of RAM. Relying on python, the software runs on Windows, macOS and Linux and it is managed through a Conda environment. Performance also depends on solver choice, mesh size and convergence settings, so users may need to optimize their system based on the specific simulations they plan to run. For this work, the software was run on a RockyLinux dedicated server, with a 32 core Intel Xeon processor and 128GB of RAM. Thread-level parallelization is not supported in QTCAD. Nevertheless, section 6.2 describes a workaround based on process-level parallelization in order to run several instances of the simulation script with different input parameters. The aim of these simulations is to study the behaviour of quantum confinement in the device over a wide range of results obtained from different simulations. Subsequently, the effects of transport can be investigated, resulting in important outputs that describes the device in a non-static condition. This work focuses primarily on the static study of the device under test, in order to fully understand its behaviour in terms of quantum confinement. Subsequently, some analyses were carried out to investigate the device behaviour at the transport level. Both are analyzed in the following section.

As reviewed before, all the simulations were performed on a dedicated server located within the university. To access it and interact with the GUI, a software called X2Go was used to establish a connection with XFCE, a lightweight desktop environment for Unix-like operating systems built on the GTK toolkit. The simulations, especially when involving large meshes, can take several minutes to complete. In this context, a problem arises. After a period of inactivity, the server stops responding to commands, making it necessary to restart the remote desktop environment. This results in the loss of work, as all processes are terminated. To address this, the simulations can be called by a terminal on a local machine via the command `ssh`. When running scripts remotely via SSH, one common issue is that the connection may be interrupted due to network instability or inactivity, as for the dedicated Linux server. If the SSH session is closed for any reason, all processes that were started within that session are usually terminated by the system. This can be particularly problematic for long-running simulations or computational tasks, which may take several hours to complete. To prevent this, it is possible to use the `nohup` command in Linux. This utility allows a command to continue running even after the terminal is closed or the SSH session is disconnected. It effectively detaches the process from the session, redirecting its standard output and standard error to a file, by default `nohup.out` if not otherwise specified.
After a SSH session is established, a long-running script can be called with

```
nohup python long_script.py > output.log 2>&1 &
```

detaching its execution from the SSH session while redirecting its standard output and error to a log file. For long simulations, it is also possible to implement an email notification system using an SMTP server and email delivery platform, such as Mailgun. A custom function, based on the `smtplib` library, can be invoked at any point in the code to send an email, providing real-time updates on the simulation progress.

```python
import smtplib
from email.mime.text import MIMEText

def send_email(subject, text, receiver):
    msg = MIMEText(text)
    msg["Subject"] = subject
    msg["From"] = email
    msg["To"] = receiver

    with smtplib.SMTP_SSL("smtp.mailgun.org", 465) as server:
        server.login(email, app_pw)
        server.send_message(msg)
    print("[EMAIL] Sent")
```

A dummy email address was created on Mailgun, along with its app password and used as the sender. It is necessary to configure a filter on the receiver account to prevent these messages from being automatically directed to the spam folder.

6.1 Device Layer Simulation

The `device` package is one of the core components of QTCAD, providing the tools needed to model realistic nanodevices through the Device class. This class serves as the main entry point for building systems where both electrostatics and quantum effects need to be captured. Nanoacademic Technologies offers a set of tutorials and practical examples to help users become familiar with the API and develop code tailored to their needs. The Device class supports the solution of Schrödinger and Poisson equations on both static and adaptive meshes, the inclusion of user-defined charge distributions and the study of band alignment, spin–orbit coupling, strain effects and valley physics. It also enables the modelling of quantum wells, donor states and quantum dots in various geometries, including FD-SOI, with extensions to many-body interactions and coupling phenomena. Visualization of simulation results is supported via ParaView, as shown in Section 6.1.6.

This package is the first layer among the three offered by QTCAD and lays the foundation for transport simulations. In this phase, a device object is defined, composed by a mesh file and some other parameters. The device can be modelled to host both electrons and holes as charge carriers and can rely on Fermi-Dirac distribution or an approximated model in order to greatly reduce simulation times. The full Fermi-Dirac distribution significantly slows down the simulator because it involves repeated evaluation of exponential functions and numerical integration of Fermi-Dirac integrals, which are computationally expensive. In contrast, common approximations (such as the Maxwell-Boltzmann limit or rational fits) replace these with simpler closed-form expressions, avoiding costly integrations and leading to much faster simulations.

After the device is created, physical properties are assigned to volumes and surfaces of the mesh, previously prepared in Gmsh. After the mesh and device are completely and correctly defined, they are fed to Poisson and Schrödinger solvers. The Poisson simulator calculates the electrostatic potential throughout the device by solving Poisson's equation, taking into account the distribution of charges, doping profiles and applied voltages. The Schrödinger simulator, on the other hand, solves the Schrödinger equation to determine the quantum states, wavefunctions and energy levels of carriers under confinement, providing insight into quantum effects such as tunnelling and discrete energy levels that are not captured by classical models. Finally, the results are processed and stored in an optimal manner.

The code will now be explained in detail.

6.1.1 Directory Organization

Before starting reviewing the code, it is necessary to understand the directory organization, schematically represented in figure 6.1.

```
simulation
├ sim.py
└ config
  ├ device.step
  ├ device.iges
  ├ device.geo
  └ meshes
    └ device.msh2
  └ images
    └ device.png
  └ exported_parameters
    └ device.csv
└ results
  ├ device_results.png
  └ device_results.mat
├ tools
└ batch_results
```

Figure 6.1: Simulation environment directory organization

The main directory contains all the simulation scripts, in this case the main script `sim.py` is shown, that executes Poisson and Schrödinger analysis. Then the directory must include the `results` folder, where all the results, mainly the image and Matlab file, are stored. The `config` folder contains all the files required for simulation:

- All the CAD files used for generating the mesh in Gmsh. The files saved here are not used in the simulations, but are placed in this folder to maintain a unified, consistent and organized project directory across all the steps preceding the QTCAD simulation.
- `meshes`: it contains all the meshes used to define the devices in the simulation script.
- `images`: each mesh file is associated with an image, which is included in the output to provide a visual representation of the device.
- `exported_parameters`: this folder includes all the parameters exported from Fusion to automatically compute dimensions for the device simulation.

It is important to note that the simulation file can handle the same type or structure of device, even if the dimensions or meshing steps differ. This means that the same device can be exported in multiple variants and then simulated by simply referring to its name in the simulation script. To achieve this, it is essential to assign the same name to the mesh, image and parameter files, while QTCAD distinguishes them based on their file extensions.

6.1.2 Environment Definition

The first step is to import all the libraries and define the input and output paths. While the relevant QTCAD API and custom libraries can be seen in the full code in the attachment B.1, the definition of the I/O paths are now reviewed.

Paths Definition

The definition of I/O paths is important to let the python script know where to take and put files, while maintaining a coherent simulation environment. The paradigm is to have the input files named the same, with different extensions. The definition of the device under test is defined through the variable `device_name`, while appending the file extensions informs the program which files to locate. The environment uses the three files described in the previous section, namely one mesh file, one image and one parameters file.

```python
# ------------------------------------------------------------------ #
# DEFINE PATHS TO INPUT AND OUTPUT FILES                             #
# ------------------------------------------------------------------ #

script_dir = pathlib.Path(__file__).parent.resolve()

# DEFINE DEVICE NAME, USED TO FIND INPUT FILES
device_name = "sqd_soi_planar"

mesh_name       = device_name + ".msh2"
image_name      = device_name + ".png"
parameters_name = device_name + ".csv"

config_dir = script_dir / "config"

path_mesh       = config_dir / "meshes" / mesh_name
path_image      = config_dir / "images" / image_name
path_parameters = config_dir / "exported_parameters" /
 ↪  parameters_name
```

Since these files are stored in their respective folders, the program requires the parent folder, named `config` and can then locate each file by searching within this folder, specifically in `meshes`, `images` and `exported_parameters`, storing the path for each file.
Then, the folder where all the results will be saved is defined, named `results`. The paths, including file names, are defined for some important output files. The `.hdf5` and `.vtu` are files generated from the simulations that stores informations about the device object and can be seen in ParaView (more on section 6.1.6), while the `path_results_img` and `path_results_mat` are used to save, respectively, an output image, containing some important plots and the simulation results in a Matlab file.

```
results_dir = script_dir / "results"

path_hdf5 = results_dir / "device_results.hdf5"
path_psi0 = results_dir / "device_psi0.vtu"
path_psi1 = results_dir / "device_psi1.vtu"

path_results_mat = results_dir / "device_results.mat"
path_results_img = results_dir / "device_results.png"
```

Parameters Definition and Geometrical Features Extraction

Next, it is important to know the physical dimensions of the device, bearing in mind that the entire setup was developed to operate in units of nanometers. The custom function `get_parameter_value` was developed to extract the value of a specific parameter from the parameters CSV file. The function is available in the attachments appendix inside the script A.8. Each value is then multiplied by $1e - 9$ to convert the value to nanometers.

```
# EXTRACT THE GEOMETRICAL DIMENSIONS FROM THE .CSV FILE EXPORTED
↪   FROM FUSION360
gate_oxide = get_parameter_value(path_parameters,
↪   "gate_oxide_thickness")*1e-9
lateral_oxide = get_parameter_value(path_parameters,
↪   "oxide_lateral")*1e-9
plunger_length = get_parameter_value(path_parameters,
↪   "plunger_gate")*1e-9
barrier_length = get_parameter_value(path_parameters,
↪   "barrier_gate")*1e-9
plunger_width = get_parameter_value(path_parameters,
↪   "channel_width")*1e-9
spacers = get_parameter_value(path_parameters, "spacers")*1e-9
contact_spacers = get_parameter_value(path_parameters,
↪   "contact_spacers")*1e-9

plunger_x_coordinate = 0
```

Here the `plunger_x_coordinate` is set to zero due to the symmetry of the device mesh and the choice to place the plunger gate at the origin, but for multidot devices it may be necessary to define multiple plunger coordinates, one for each dot. This can be done automatically, as shown for the double quantum dot example in Section 6.1.3.
Then some variables are defined to store informations about the device temperature, materials, dopings and voltages. The package `mt` is an alias for the `materials` library, from `qtcad.device`. Storing the semiconductor and dielectric material choice inside a variable enables the simulation of different materials later on. As said before, QTCAD

does not support metal contacts. Instead it simulates the interface between a metal and other surfaces by applying a specific type of boundary condition and the desired metal workfunction, expressed in Joules.

```
# DEFINE THE DEVICE WORKING TEMPERATURE IN KELVIN
device_temperature = 0.01

# DEFINE PHYSICAL MATERIALS
semiconductor = mt.Si       # Silicon as semiconductor
dielectric    = mt.SiO2     # Silicon dioxide as the dielectric
metal_workfunction = 4.33 * ct.e     # Titanium workfunction (Joules)

# DEFINE DOPING AND VOLTAGES
n_doping  = 1e18*1e6
V_plunger = 0.5
V_barrier_1 = -0.1
V_barrier_2 = -0.1
```

Then, the backgate properties are defined. Here, it is assumed that the semiconductor below the buried oxide, which is in contact with the back gate, is n-doped silicon with a dopant concentration of $10^{15} cm^{-3}$. In addition, a dopant ionization energy of $46 meV$ is used, which is appropriate for phosphorus donors.

```
# DEFINE BACKGATE PROPERTIES
use_backgate = False  # Backgate only defined if this is set True
V_backgate = -0.5
backgate_doping = "n"
backgate_dose = 1e15*1e6
backgate_binding_energy = 46e-3*ct.e  # Dopant ionization energy for
 ↪  phosphorus donors
```

Next, some boolean flag variables are defined to control whether the HDF5 and VTU files are saved. Since these files are large and can take considerable time to export, setting the corresponding flag allows them to be generated only when needed.

```
# CHOOSE IF SAVING THE HDF5 FILE, USED FOR THE TRANSPORT LAYER
 ↪  SIMULATION
save_hdf5 = True

# CHOOSE IF SAVING THE EIGENSTATES IN .VTU FORMAT
save_psi0_vtu = True
save_psi1_vtu = False
```

Mesh Loading and Device Initialization

Now the mesh can be loaded into a `mesh` object with the relative scaling of $1e-9$, to set the unit to nanometers. With the command `glob_nodes` it is possible to retrieve the coordinates of each node of the mesh. Then by summing the absolute value of the minimum and the maximum vale of those, thanks to the device symmetry, the full device dimensions is calculated for each axis. To get the device length, here stored in the variable `device_length`, the total dimension in the x axis is subtracted from the two lateral shallow trench isolation oxide barriers. Keep in mind that the `dim_` variables are not in nanometers.

```
# SET MESH SCALING FACTOR TO NANOMETERS
scaling = 1e-9

# DEFINE MESH VIA SCALING FACTOR AND MESH PATH
mesh = Mesh(scaling, path_mesh)

# GLOBAL NODES OF FULL DEVICE
x = mesh.glob_nodes[:, 0]
y = mesh.glob_nodes[:, 1]
z = mesh.glob_nodes[:, 2]

# FULL DIMENSIONS OF THE DEVICE IN NANOMETERS
dim_x = np.abs(np.min(x)*1e9)+np.abs(np.max(x)*1e9)
dim_y = np.abs(np.min(y)*1e9)+np.abs(np.max(y)*1e9)
dim_z = np.abs(np.min(z)*1e9)+np.abs(np.max(z)*1e9)

# DEFINE THE DEVICE LENGTH AS THE TOTAL LENGTH MINUS THE LATERAL
↪  OXIDE
device_length = dim_x*1e-9 - 2*lateral_oxide
```

Now the Device object can be instantiated, linking it to the mesh and setting the confined carries to electrons. It follows that the implied statistic distribution is the Fermi-Dirac one. Here we use the approximated one to save simulation time. The device temperature is also set.

```
# CREATE DEVICE FROM MESH AND SET CONFINED CARRIERS TO ELECTRONS
d = Device(mesh, conf_carriers = "e")
d.set_temperature(device_temperature)
d.statistics = "FD_approx"  # Aproximated Fermi-Dirac distribution
```

6.1.3 Assigning Physical properties

The assignment of physical properties allows the simulator to understand what part of the mesh corresponds to what material or contact. The labels defined in Gmsh are used to make this associations. The volumes are assigned to a specific material and surfaces to a specific boundary condition.

To specify a volume material the command `new_region` is used, taking as argument the volume label as well as the material, previously defined in distinct variables for the semiconductor and the dielectric. For the source and drain volumes, the doping concentration is defined via the two parameters `pdoping` and `ndoping`. In Section 7.4 it is shown how to include different volumes for source and drain to simulate the effects of doping diffusion gradients. The key idea is to provide a good ohmic contact for the electrons where tunnel from, acting as a local carrier reservoir. The channel is kept intrinsic to minimize noise arising from crystal imperfections. If doping were introduced, free carriers would be present, which are likely to have poor confinement due to their relatively high energy, even at low temperatures, making it difficult to localize them in a quantum dot and potentially reducing coherence times. Additionally, the dopant-induced electric fields could perturb the confinement potential, introducing local dipoles due to differences in electronegativity and atomic number, thereby increasing the risk of quantum decoherence and shortening qubit lifetimes.

```
# DEFINE VOLUME PHYSICAL CONDITIONS
d.new_region("semi_channel", semiconductor)

d.new_region("nsemi_source", semiconductor, pdoping=0,
↪   ndoping=n_doping)
d.new_region("nsemi_drain", semiconductor, pdoping=0,
↪   ndoping=n_doping)

d.new_region("diel_oxide",     dielectric)
d.new_region("diel_barrier_1", dielectric)
d.new_region("diel_barrier_2", dielectric)
d.new_region("diel_plunger",   dielectric)
```

Then, the surfaces boundary conditions are applied, to simulate the effects of contacts. The metal-semiconductor leads form an ohmic contact, instantiated by the command `new_ohmic_bnd` specifying the name of the label used in Gmsh. Then, for the metal-oxide gate contacts it is necessary to use the `new_gate_bnd` command, that takes as arguments also the applied potential and the metal workfunction it is needed to simulate.

```
# DEFINE SURFACES BOUNDARY CONDITIONS
d.new_ohmic_bnd("ohmicbnd_drain")
d.new_ohmic_bnd("ohmicbnd_source")

d.new_gate_bnd("gatebnd_barrier_1", V_barrier_1, metal_workfunction)
d.new_gate_bnd("gatebnd_barrier_2", V_barrier_2, metal_workfunction)
d.new_gate_bnd("gatebnd_plunger",   V_plunger_1, metal_workfunction)
```

If the boolean flag for the use of the backgate is set to `True`, a new boundary condition is instantiated for the backgate contact. A Frozen boundary condition is applied, which is suitable for ohmic contacts in weakly doped semiconductors at cryogenic temperatures. This choice ensures that the potential at the backgate remains fixed during the simulation, reflecting the behaviour of a real backgate contact in UTBB technologies.

```
if use_backgate:
  d.new_frozen_bnd("gatebnd_backgate", V_backgate, semiconductor,
   ↪  backgate_dose, backgate_doping, backgate_binding_energy)
```

Then, it is necessary to define a list of regions, called `dot_region`, where no classical charge is allowed. The regions include those that accommodate the quantum dot, like the channel, as well as all other regions that forms a bandgap difference barrier, such as the oxide. If a region is not included in this array, it will not be taken into account for Schrödinger simulation. Finally, the dot region is passed to the device object via the `set_dot_region` command.

```
# DEFINE THE DOT REGION AS A LIST OF REGION LABELS THAT COMPOSE THE
↪  DOT MEDIUM AND BARRIERS
dot_region = ["semi_channel", "diel_oxide", "diel_barrier_1",
↪  "diel_barrier_2", "diel_plunger"]

# SET UP THE DOT REGION IN WHICH NO CLASSICAL CHARGE IS ALLOWED
d.set_dot_region(dot_region)
```

The device is now ready to be fed to the Poisson and Schrödinger solvers.

Extension for Double Quantum Dot Devices

Some modifications can be introduced in order to adapt the code for multidot devices. Firstly, some physical dimensions can be calculated automatically from the exported parameters. In this example, the x coordinate of the plunger gate was defined for the one near the drain. In a multidot device, each plunger gate coordinate can be defined to perform analysis on each of them. Follows an example for a double quantum dot configuration.

```
# COMPUTE SOME DIMENSIONS FROM EXTRACTED PARAMETERS
contact_dot_spacing = spacers + contact_spacers + barrier_length +
↪  plunger_length/2
opposite_contact_dot_spacing = 3*spacers + contact_spacers +
↪  2*barrier_length + plunger_length*3/2
dot_spacing = barrier_length + 2*spacers + plunger_length
plunger_x_coordinate = (barrier_length + plunger_length)/2 + spacers
```

Then, the voltages for the additional gates are defined and the physical properties are updated to accommodate the changes, including them in the `dot_region` variable.

```
n_doping  = 1e18*1e6
V_plunger_1 = 0.5
V_plunger_2 = 0.5
V_barrier_1 = -0.1
V_barrier_2 = -0.2
V_barrier_3 = -0.1
```

```
d.new_region("diel_barrier_1", dielectric)
d.new_region("diel_barrier_2", dielectric)
d.new_region("diel_barrier_3", dielectric)
d.new_region("diel_plunger_1", dielectric)
d.new_region("diel_plunger_2", dielectric)

d.new_gate_bnd("gatebnd_barrier_1", V_barrier_1, metal_workfunction)
d.new_gate_bnd("gatebnd_barrier_2", V_barrier_2, metal_workfunction)
d.new_gate_bnd("gatebnd_barrier_3", V_barrier_3, metal_workfunction)
d.new_gate_bnd("gatebnd_plunger_1", V_plunger_1, metal_workfunction)
d.new_gate_bnd("gatebnd_plunger_2", V_plunger_2, metal_workfunction)
```

```
dot_region = ["semi_channel", "diel_oxide", "diel_barrier_1",
↪  "diel_barrier_2", "diel_barrier_3", "diel_plunger_1",
↪  "diel_plunger_2"]
```

6.1.4 Poisson and Schrödinger Solvers

Now that the device object is completed, it is possible to define a `PoissonSolver` object with its relative `PoissonSolverParams`. The only set parameter is the tolerance between two successive Poisson iterations, in volts. QTCAD supports an adaptive Poisson solver, that can work with a `.geo_unrolled` file. This derives from Gmsh and contains all the informations required to do a local meshing of a geometry. By providing the solver with a coarse-grained mesh along with its corresponding `.geo_unrolled` file, it is possible to perform a meshing algorithm on the geometry if the Poisson iterations fail to converge after a certain number of loops. This approach allows the use of coarse meshes, saving time in the meshing step in Gmsh and saving space on the hard drive. Unfortunately, this method seems to work only for geometries defined inside Gmsh with its proprietary `.geo` compatible commands. Importing a geometry from an external CAD software could trigger errors during the local meshing step, indicating that some curves do not close properly. This is likely caused by the use of the `Coherence` command, which was applied to make the geometry conformal. The use of the adaptive Poisson solver is the preferred choice. However, some adjustments are required to adapt the workflow and prevent errors during the local meshing step. Due to these issues, the standard non-linear Poisson solver was implied, which simply iterates until the error falls below the specified tolerance.
After the simulation is done, the results can be stored inside the HDF5 file, if the corresponding boolean flag is set to `True`.

```
# ---------------------------------------------------------------- #
# NON-LINEAR POISSON SOLVER                                        #
# ---------------------------------------------------------------- #

# CREATE A POISSON SOLVER PARAMETERS OBJECT
params_poisson = PoissonSolverParams()

# The tolerance attribute tol specifies the maximum acceptable
↪  potential difference (in Volts) between two successive
↪  self-consistent-loop iterations
params_poisson.tol = 1e-5

# CREATE AND SOLVE A NON-LINEAR POISSON SOLVER
s = PoissonSolver(d, solver_params=params_poisson)
s.solve()

# SAVE POISSON RESULTS IN THE HDF5 FILE
if save_hdf5:
    io.save(str(path_hdf5), {"n": d.n/1e6, "p": d.p/1e6, "phi":
    ↪  d.phi, "EC": d.cond_band_edge()/ct.e, "EV":
    ↪  d.vlnce_band_edge()/ct.e})
```

As for the Poisson solver, the Schrödinger solver is defined with its tolerance parameter. Here, the potential landscape is firstly set from Poisson with the command `set_V_from_phi`. To start a Schrödinger simulation, it is necessary to not use the same device object as before, but a SubDevice and a SubMesh, composed only by the `dot_region` defined for the mesh. For this example, only the source and drain contacts were excluded from the simulation.
After the simulation ends, the eigenenergies are displayed.

```
# ---------------------------------------------------------------- #
# SCHRODINGER SOLVER                                               #
# ---------------------------------------------------------------- #

# GET THE POTENTIAL ENERGY FROM THE BAND EDGE FOR USAGE IN THE
↪  SCHRODINGER SOLVER
d.set_V_from_phi()

# CREATE A SUBMESH INCLUDING ONLY THE DOT REGION AND A SUBDEVICE FOR
↪  THE LATTER
submesh = SubMesh(d.mesh, dot_region)
subdevice = SubDevice(d, submesh)

# CREATE A SCHRODINGER SOLVER PARAMETERS OBJECT
params_schrod = SchrodingerSolverParams()
params_schrod.num_states = 4     # Specify the number of eigenstates
↪  and energies to consider in the diagonalization of the dot
↪  Hamiltonian
params_schrod.tol = 1e-12     # Set the tolerance for convergence on
↪  energies in electron-volts

# CREATE AND SOLVE A SCHRODINGER SOLVER
schrod_solver = SchrodingerSolver(subdevice)
schrod_solver.solve()

# PRINT EIGENENERGIES
subdevice.print_energies()
```

Now the main simulation steps are done. The results are now ready to be processed to extract the information of interest. The next section addresses these step in details.

6.1.5 Results Display

Now that the simulation is complete, it is necessary to correctly process and plot the results, in order to get the most out of those. The first step is to define the dimensions of the subdevice used for Schrödinger simulation. This is done as before using `glob_nodes` of the submesh linked to the subdevice.

```
# GLOBAL NODES FOR THE SUBDEVICE
xdot = submesh.glob_nodes[:, 0]
ydot = submesh.glob_nodes[:, 1]
zdot = submesh.glob_nodes[:, 2]
```

In order to place linecuts at exactly the dot location it is needed to find the max probability amplitude of the z axis, known that the device is fully symmetrical on the y axis and so setting the y coordinate to zero.

```
# START BY FINDING MAX PROBABILITY FOR Z AXIS UNDER THE PLUNGER GATE
# THEN PLACE X AND Y LINECUTS AT THAT Z COORDINATE
psi0 = np.abs(subdevice.eigenfunctions[:, 0])**2
```

With `psi0` found, it is possible to perform a linecut in the ZoY plane by setting the starting point from the top of the silicon, at the interface with the gate oxide and the end at the bottom, at the interface with the BOX. Both point are fixed at the plunger gate x coordinate, which in the case of a single quantum dot modelled symmetrical in each direction, corresponds to zero. This is done for both the ground state eigenfunction and the applied potential, used for plotting them in conjunction. Then, for plotting consistency, the two results `psiposz` (a function of `psiz0`) and `V_plungerosz` (a function of `Vz`) are inverted and shifted by their maximum value in order to align them with the gate oxide interface, where the zero of the z axis is defined. Then the z coordinate of the peak of the wavefunction is obtained by taking the `psiposz` value at maximum `psiz0`.

```
# WAVEFUNCTION AND POTENTIAL ENERGY IN THE ZoY PLANE
beginz = (plunger_x_coordinate, 0, np.max(zdot))
endz =   (plunger_x_coordinate, 0, np.min(zdot))
psiposz, psiz0 = linecut(submesh, psi0, beginz, endz)
psiposz = -psiposz
psiposz += np.max(zdot)
V_plungerosz, Vz = linecut(mesh, d.V, beginz, endz)
V_plungerosz = -V_plungerosz
V_plungerosz += np.max(z)

# FIND MAXIMA OF PSI0_Z AND PSI0_X
dot_z_position = psiposz[np.argmax(psiz0)]
```

Now that the z position of the dot is known it is possible to proceed with the definition of the linecuts of the XoZ and XoY planes, both by fixing the z coordinate to the dot location just calculated. In the first case the x axis sweeps from the center to the two sides with magnitude `device_length/2`, while in the second case the x coordinate is fixed to the plunger coordinate and the y axis sweeps to the maximum and minimum node, corresponding to the top of the gate oxide and bottom of BOX (where the backgate is localized).

```
# WAVEFUNCTION AND POTENTIAL ENERGY IN THE XoZ PLANE
beginx = (-device_length/2, 0, dot_z_position)
endx =   (device_length/2, 0, dot_z_position)
psiposx, psix0 = linecut(submesh, psi0, beginx, endx)
psiposx += (np.min(xdot) + lateral_oxide)
V_plungerosx, Vx = linecut(mesh, d.V, beginx, endx)
V_plungerosx += (np.min(x) + lateral_oxide)

# WAVEFUNCTION AND POTENTIAL ENERGY IN THE YoX PLANE
beginy = (plunger_x_coordinate, np.min(y), dot_z_position)
endy =   (plunger_x_coordinate, np.max(y), dot_z_position)
psiposy, psiy0 = linecut(submesh, psi0, beginy, endy)
psiposy += np.min(ydot)
V_plungerosy, Vy = linecut(mesh, d.V, beginy, endy)
V_plungerosy += np.min(y)
```

Along with the wavefunction, the potential landscape is found as before. This is done to plot the two quantities together and see how the potential affects the bending of the conduction band and so the wavefunction localization. One of the first things to notice is how local minima in the conduction band lead to local maxima in the wavefunction.

Now a simple code follows, used to determine if the dot is effectively confined under the plunger gate. The wavefunction along x is firstly normalized and then cropped around the plunger gate. To do so a custom function called `crop_around_coordinate` is used, cutting the values of `normalized_psix0` around the plunger gate with length `plunger_length`. The function is available in the attachments appendix inside the script A.6. Then, the maximum vale of this subsection of the wavefunction is found. This is done to find for the local maximum near the plunger gate, useful when the device is multidot. A custom function called `is_within_percentage` is used to check if the x position of the peak of the wavefunction is in the neighbourhood of the plunger gate center position with a defined deviation threshold percentage, in this case 20%. The function is available in the attachments appendix inside the script A.5. It returns a boolean, so can be placed inside an `if` statement to directly print on the terminal if the dot is localized under the plunger gate or not.

```
# FIND MAXIMA OF PSIO_X
normalized_psix0 = psix0/np.max(psix0)

dot1_x, dot1_psix0 = crop_around_coordinate(psiposx,
↪  normalized_psix0, np.abs(plunger_x_coordinate), plunger_length)

dot1_x_position = dot1_x[np.argmax(dot1_psix0)]

localized_dot=False
if is_within_percentage(np.abs(dot1_x_position*1e9),
↪  np.abs(plunger_x_coordinate*1e9), 20):
  localized_dot=True

if localized_dot:
  print("[OK] Dot is localized under the plunger gate!")
else:
  print("[KO] Dot IS NOT localized under the plunger gate!")
```

Then, additional informations can be printed regarding the confinement along the z and x coordinates before the export of the final images. Firstly, the z and x locations of the dot are printed. Then the function `x_for_threshold` is used, as for the previous one, available in the script A.6. It calculates the ratio between the area of the wavefunction cropped around the plunger gate and the area of the total wavefunction. This is done to calculate how much the dot is localized under the plunger gate. For a single quantum dot, the best possible result is 100% (dot completely constrained under the plunger gate).

```
print(f"Quantum dots found at z = {dot_z_position*1e9:.2f} nm")

print(f"Quantum dot found at x = {dot1_x_position*1e9:.2f} nm vs
↪  plunger at {np.abs(plunger_x_coordinate*1e9):.2f} nm")

dot1_x_confinement = x_for_threshold(psiposx, normalized_psix0,
↪  -np.abs(plunger_x_coordinate), plunger_length)

print(f"Quantum dot confinement = {dot1_x_confinement*100:.2f}%")
```

The next step is to calculate and print the cumulative confinement levels associated to the z-axis probability density function. In this work, the $N\sigma$ notation is used to indicate the confinement level corresponding to a cumulative probability containing N nines, from $2\sigma = 99\%$ to $6\sigma = 99.9999\%$. To do so, a function called `z_for_threshold` is used, available in the attachments appendix inside the script A.3. It takes as input the z coordinate (in nanometers), the normalized probability density function along that axis and a threshold factor. It calculates where the integral of the normalized probability density function starting from the beginning of the z axis reach the threshold parameter. By calling the function for different thresholds of number of nines it is possible to calculate and plot all the z coordinates where the dot is localized from 99% to 99.9999%.

```
# COMPUTE Z-AXIS CUMULATIVE CONFINEMENT LEVELS
z_thresh_2N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
 ↪  threshold=0.99)*1e-9
z_thresh_3N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
 ↪  threshold=0.999)*1e-9
z_thresh_4N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
 ↪  threshold=0.9999)*1e-9
z_thresh_5N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
 ↪  threshold=0.99999)*1e-9
z_thresh_6N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
 ↪  threshold=0.999999)*1e-9
print(f"2N threshold for quantum dot at z = {z_thresh_2N*1e9:.2f}
 ↪  nm")
print(f"3N threshold for quantum dot at z = {z_thresh_3N*1e9:.2f}
 ↪  nm")
print(f"4N threshold for quantum dot at z = {z_thresh_4N*1e9:.2f}
 ↪  nm")
print(f"5N threshold for quantum dot at z = {z_thresh_5N*1e9:.2f}
 ↪  nm")
print(f"6N threshold for quantum dot at z = {z_thresh_6N*1e9:.2f}
 ↪  nm")
```

Matlab amd VTU Export

Now that all the information that can be printed in the terminal has been displayed, it is possible to proceed with saving all the results into a MATLAB file. This is useful as all the informations can be displayed and processed inside Matlab, without the need to perform the simulation again later on. The file has a `.mat` extension and there is a dedicated function called `savemat` for formatting and saving a Matlab file inside the library `scipy.io`. Here it is possible to include all the required variables. For convenience, all positions and dimensions are normalized to 10^0, removing the nanometer scale in order to speed up execution in MATLAB and enhance numerical accuracy.
The function needs a file path and a list of couples `key:value`, in order to save the data correctly. Later in Matlab, the values (for example `psi0_x`) will be accessible by using the command:

```
data = load('device_results.mat');
psix0 = data.psi0_x;
```

```
# SAVE DEVICE RESULTS IN THE .MAT FILE
savemat(path_results_mat, {
"x":psiposx/1e-9,
"y":psiposy/1e-9,
"z":psiposz/1e-9,
"psi0_x":psix0,
"psi0_y":psiy0,
"psi0_z":psiz0,
"V_x":Vx/ct.e,
"V_y":Vy/ct.e,
"V_z":Vz/ct.e,
"doping":n_doping,
"V_plunger":V_plunger,
"V_barrier":V_barrier,
"dot_z":dot_z_position/1e9,
"dot_z_threshold_2N":z_thresh_2N/1e9,
"dot_z_threshold_3N":z_thresh_3N/1e9,
"dot_z_threshold_4N":z_thresh_4N/1e9,
"dot_z_threshold_5N":z_thresh_5N/1e9,
"dot_z_threshold_6N":z_thresh_6N/1e9,
"energies":subdevice.energies/ct.e
})
```

Furthermore, it is possible to export the ground state and first excited state (or more if needed) inside a dedicated `.vtu` file to be seen later in ParaView (more on this topic in Section 6.1.6). These files are large and require significant storage space. Therefore, saving is enabled only when needed, by setting the associated boolean option variables to `True` in the script header.

```
# SAVE SCHRODINGER RESULTS IN .VTU FORMAT
if save_psi0_vtu:
  io.save(path_psi0, np.abs(subdevice.eigenfunctions[:, 0])**2,
   ↪  submesh)
if save_psi1_vtu:
  io.save(path_psi1, np.abs(subdevice.eigenfunctions[:, 1])**2,
   ↪  submesh)
```

Image Export

It is now possible to export plots as images to better illustrate the confinement behaviour. To provide a clear and structured understanding of the device, a single composite image was generated, arranged into four quadrants, each containing different informations. Firstly, some text is declared. Mainly to display informations about the mesh, like node number and physical dimensions. In the script this is called `title`. Then the second and third texts, named `text_vars` and `text_desc` respectively, contains general informations about the simulation variables and device results. The last text `text_conf` contains information about the z-axis cumulative confinement levels, as described in the previous pages. In the text `text_desc` the variables `leverarm` and `score` were retained in this example, although its introduction will be discussed later in Section 6.3.1 and 6.2.1, respectively. In the text `text_vars` a custom function called `get_apex` is used to display scientific notation, available in the attachments appendix as script A.7.

```
title = (
fr'{mesh_name}' + '\n' +
fr'{mesh.node_number} nodes, {dim_x:.1f}x{dim_y:.1f}x{dim_z:.1f}
↪  nm'
)
text_vars = (
fr'T = {device_temperature:.2f} K' + '\n' +
fr'$dose_{{\mathrm{{n}}}}$ = ' + get_apex(n_doping) + '
↪  $m^{{\mathrm{{-3}}}}$' + '\n' +
fr'$V_{{\mathrm{{plunger}}}}$ = {V_plunger:.2f} V' + '\n' +
fr'$V_{{\mathrm{{b\_lateral}}}}$ = {V_barrier_1:.2f} V' + '\n' +
fr'$V_{{\mathrm{{b\_central}}}}$ = {V_barrier_2:.2f} V'
)

text_desc=(
fr'$z_{{\mathrm{{dot}}}}$ = {dot_z_position*1e9:.2f} nm' + '\n' +
fr'2$\sigma$ = {z_thresh_2N*1e9:.2f} nm' + '\n' +
fr'$\alpha$ = {leverarm*1000:.1f} meV/V' + '\n' +
fr'score = {int(score)} pt'
)

text_conf=(
fr'$z_{{\mathrm{{dot}}}}$ = {dot_z_position*1e9:.2f} nm' + '\n' +
fr'2$\sigma$ = {z_thresh_2N*1e9:.2f} nm' + '\n' +
fr'3$\sigma$ = {z_thresh_3N*1e9:.2f} nm' + '\n' +
fr'4$\sigma$ = {z_thresh_4N*1e9:.2f} nm' + '\n' +
fr'5$\sigma$ = {z_thresh_5N*1e9:.2f} nm' + '\n' +
fr'6$\sigma$ = {z_thresh_6N*1e9:.2f} nm'
)
```

Now that the text is ready, it is possible to define dimensions and DPI (Dots Per Inch) resolution for the image. By choosing the 2x2 format, as shown in the code, the figure can be created with the command `plt.figure`, where `plt` is an alias for the function `pyplot` inside the library `matplotlib`.

```
# DPI AND ORIGINAL DIMENSIONS FOR A SINGLE PLOT
dpi = 300
single_w_px = 1500
single_h_px = 1000

# 2 COLUMNS x 2 ROWS
fig_w_px = single_w_px * 2
fig_h_px = single_h_px * 2

fig = plt.figure(figsize=(fig_w_px/dpi, fig_h_px/dpi), dpi=dpi)

# SPACING
w_frac = 0.36
h_frac = 0.375
```

Next, a variable `text_loc` is defined. Later this will be set as the prefix of the image file name, where if the dot, interpreted as the peak of the x-axis wavefunction, is localized under the plunger gate the prefix will be the score of the configuration (more on section 6.2.1), while if the dot is not localized the prefix will be an X, identifying a wrong configuration.

```
text_loc="X"
if localized_dot:
  text_loc=f"{int(score)}"
```

The first quadrant of the figure is then filled with the mesh name and image, defined in the paths at the beginning of the script. All the informations included in the three text variables defined before are displayed. It is important to keep in mind that the quadrant order follows a row by row and column by column ordering, so the first quadrant is the upper left, the second the upper right, the third the bottom left and the fourth the bottom right.

```
# QUADRANT 1: DEVICE INFORMATIONS AND SIMULATION VARIABLES
ax_text = fig.add_axes([0.02, 0.52, w_frac, h_frac])
ax_text.axis('off')
ax_text.text(0.5, 0.95, title, fontsize=15, ha='center', va='center',
↪   wrap=True)
ax_text.text(-0.025, 0.15, textstr, fontsize=14, ha='left',
↪   va='center', wrap=True)
ax_text.text(0.625, 0.15, text_desc, fontsize=14, ha='left',
↪   va='center', wrap=True)
img = mpimg.imread(path_image)
imagebox = OffsetImage(img, zoom=0.1)
ab = AnnotationBbox(imagebox, (0.5, 0.65), frameon=False,
↪   xycoords='axes fraction')
ax_text.add_artist(ab)
```

Then the second quadrant can be defined. It includes the plots of the x-axis wavefunction and potential. Some key results were added in order to get information about the x-axis confinement goodness. The two main informations added here are:

- The percentage of confinement of the dot under the plunger gate. This is shown in a blue box and it was computed before with the `x_for_threshold` function and stored in the variable `dot1_x_confinement`. This parameter can be computed for each dot of the device. In Section 6.1.5 an example for a double quantum dot is shown.
- The x coordinate of the peak of the wavefunction. This is shown in a green box and it was computed before with the `crop_around_coordinate` function and stored in the variable `dot1_x_position`. This was the same variable used to verify that the dot is localized under the plunger gate. Although the device is fully symmetrical, in some cases, particularly for multidot devices, the dot position along the x axis may deviate. Ideally, this value should coincide with the plunger gate location, namely with its center along the x axis.

Also, in the title of the plot the z coordinate of the dot is displayed.

```
# QUADRANT 2: LINECUT ALONG CHANNEL
y_percentage_label_coordinate = 0.5
ax1 = fig.add_axes([0.52, 0.52, w_frac, h_frac])
ax2 = ax1.twinx()
ax1.set_title(fr'Linecut along channel ($x$) @ $z_{{\mathrm{{dot}}}}
↪   = {dot_z_position*1e9:.2f}\,\mathrm{{nm}}$')
ax1.plot(psiposx / 1e-9, psix0/np.max(psix0), linewidth=2)
ax2.plot(V_plungerosx / 1e-9, Vx / ct.e, '--r')
ax1.grid()
ax1.set_xlabel("x [nm]")
ax1.set_ylabel(r"Normalized $|\Psi(x, 0, z_{dot})|^2 [m^{-3}]$",
↪   color='blue')
ax2.set_ylabel(r"$V [eV]$", color='red')
for label in ax1.get_yticklabels():
  label.set_color('blue')
for label in ax2.get_yticklabels():
  label.set_color('red')
# Add confinement percentage and peak x location on each dot
ax2.text(((-np.abs(dot1_x_position)+device_length/2)/device_length),
↪   y_percentage_label_coordinate, f"{dot1_x_confinement*100:.1f}%",
         transform=ax1.transAxes,
         fontsize=10,
         zorder=10,
         verticalalignment='center',
         horizontalalignment='center',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white',
          ↪   alpha=0.8, edgecolor='blue'))
ax2.text(((dot1_x_position+device_length/2)/device_length),
↪   np.max(dot1_psix0)-0.05, f"{dot1_x_position*1e9:.1f}nm",
         transform=ax1.transAxes,
         fontsize=10,
         zorder=10,
         verticalalignment='center',
         horizontalalignment='center',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white',
          ↪   alpha=0.8, edgecolor='green'))
```

For the third quadrant, only the wavefunction and potential along y axis is shown. The linecut should be placed under a plunger gate so that the wavefunction correspond to a dot. In a multidot device it is possible to plot one or more by modifying the code.

```
# QUADRANT 3: LINECUT ALONG Y
ax3 = fig.add_axes([0.02, 0.02, w_frac, h_frac])
ax4 = ax3.twinx()
ax3.set_title(fr'Linecut along $y$ @ $z_{{\mathrm{{dot}}}}$,
↪  $x_{{\mathrm{{plunger}}}}$')
ax3.plot(psiposy / 1e-9, psiy0/np.max(psiy0), linewidth=2)
ax4.plot(V_plungerosy / 1e-9, Vy / ct.e, '--r')
ax3.grid()
ax3.set_xlabel("y [nm]")
ax3.set_ylabel(r"Normalized $|\Psi(x_{plunger} , y, z_{dot})|^2
↪  [m^{-3}]$", color='blue')
ax4.set_ylabel(r"$V [eV]$", color='red')
for label in ax3.get_yticklabels():
  label.set_color('blue')
for label in ax4.get_yticklabels():
  label.set_color('red')
```

The fourth and final quadrant is then defined, showing the wavefunction and potential along the z axis. In this case, the values were previously reversed and shifted to ensure consistency with each other and with the axis reference. Since the mesh places the origin of all axes, including the z axis, at the semiconductor/oxide interface, this point corresponds to 0 in the plot. Positive values represent the potential within the oxide, while for negative values the effect of the applied plunger voltage produces an energy band bending that leads to the localization of the probability density function at the interface.
Then, the text containing information about the z-axis confinement, with the different values of sigma, is placed in the top-left corner. As for quadrant two, the title indicates where the linecut was done, in this case the plunger x coordinate contained in the variable `plunger_x_coordinate`.

```
# QUADRANT 4: LINECUT ALONG Z
ax5 = fig.add_axes([0.52, 0.02, w_frac, h_frac])
ax6 = ax5.twinx()
ax5.set_title(fr'Linecut along $z$ @ $x_{{\mathrm{{plunger}}}} =
↪  {plunger_x_coordinate*1e9:.2f}\,\mathrm{{nm}}$')
ax5.plot(psiposz / 1e-9, psiz0/np.max(psiz0), linewidth=2)
ax6.plot(V_plungerosz / 1e-9, Vz / ct.e, '--r')
ax5.grid()
ax5.set_xlabel("z [nm]")
ax5.set_ylabel(r"Normalized $|\Psi(x_{plunger} , 0, z)|^2 [m^{-3}]$",
↪  color='blue')
ax6.set_ylabel(r"$V [eV]$", color='red')
for label in ax5.get_yticklabels():
  label.set_color('blue')
for label in ax6.get_yticklabels():
  label.set_color('red')
ax6.text(0.02, 0.98, text_conf,
         transform=ax6.transAxes,
         fontsize=11,
         zorder=10,
         verticalalignment='top',
         horizontalalignment='left',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white',
          ↪  alpha=0.7, edgecolor='none'))
```

Finally, the image can be saved with the previously defined path and DPI.

```
# SAVE IMAGE
plt.savefig(path_results_img, dpi=dpi, bbox_inches='tight')
```

The simulation via the `device` package is now complete.

Double Quantum Dot Results

The code can be modified to host a multidot configuration and, knowing the target location of each quantum dot, it is possible to extract important parameters that not only describes the quantum confinement for each one but also the interdot coupling and interferences of gates. For this work, it was important to study the behaviour of double quantum dot devices as a starting point to use one as a spin qubit and the other one as a spin-to-charge conversion dot for Pauli spin blockade readout, analyzed in Section 2.6.3.
In a DQD the same results can be extracted as for the first one, like the position and confinement with respect to the plunger area.

```
dot1_x, dot1_psix0 = crop_around_coordinate(psiposx,
↪  normalized_psix0, -np.abs(plunger_x_coordinate), plunger_length)
dot2_x, dot2_psix0 = crop_around_coordinate(psiposx,
↪  normalized_psix0, np.abs(plunger_x_coordinate), plunger_length)

dot1_x_position = dot1_x[np.argmax(dot1_psix0)]
dot2_x_position = dot2_x[np.argmax(dot2_psix0)]

dot1_x_confinement = x_for_threshold(psiposx, normalized_psix0,
↪  -np.abs(plunger_x_coordinate), plunger_length)
dot2_x_confinement = x_for_threshold(psiposx, normalized_psix0,
↪  np.abs(plunger_x_coordinate), plunger_length)
```

The localization of the dot under a plunger gate, with the exploit of the functions `is_within_percentage`, can be done for both the dots. Then it is possible to display the informations about the second dot. By finding the difference between the two confinement areas, found integrating under the plunger area the wavefunction cropped for each dot, it is possible to display the difference (or delta) of confinement.

```
print(f"1st quantum dot found at x = {dot1_x_position*1e9:.2f} nm vs
↪  plunger at {-np.abs(plunger_x_coordinate*1e9):.2f} nm")
print(f"2nd quantum dot found at x = {dot2_x_position*1e9:.2f} nm vs
↪  plunger at {np.abs(plunger_x_coordinate*1e9):.2f} nm")

print(f"1st quantum dot confinement = {dot1_x_confinement*100:.2f}%
↪  (DQD) - {dot1_x_confinement*200:.2f}% (SQD)")
print(f"2nd quantum dot confinement = {dot2_x_confinement*100:.2f}%
↪  (DQD) - {dot2_x_confinement*200:.2f}% (SQD)")
print(f"Delta confinement = {np.abs((dot1_x_confinement/0.5) -
↪  (dot2_x_confinement/0.5))*500:.2f}%")
```

In the best scenario, the two dots are localized at the exact same coordinate with respect to a common symmetry axis, same z coordinate for the peak of the wavefunction and same area. But, the eigenstates are not localized in one dot but form symmetric (bonding) and antisymmetric (antibonding) superpositions spread over both dots. Any asymmetry, due to disorder, gate fluctuations, or even numerical imperfections, breaks this balance and makes the wavefunction appear more localized in one dot. It is important to include the asymmetry analisis for multidot configurations.

Now it is possible to plot these informations in the second quadrant of the image. The other plots, in the third and fourth quadrants, remain unchanged. The linecuts defined for these plots are taken at a specific x coordinate, namely at the center of the plunger gate. It is possible to investigate confinement along the y and z axes by selecting one of the available plunger gates, or by modifying the code to plot all of them. However, confinement along these axes usually remains the same, so this step is not strictly necessary. For the x-axis confinement plot, two additional textboxes are included to display the x-coordinate of the wavefunction peak and the confinement area under the relative plunger gate.

```
ax2.text(((np.abs(dot2_x_position)+device_length/2)/device_length),
↪   y_percentage_label_coordinate, f"{dot2_x_confinement*100:.1f}%",
         transform=ax1.transAxes,
         fontsize=10,
         zorder=10,
         verticalalignment='center',
         horizontalalignment='center',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white',
          ↪   alpha=0.8, edgecolor='blue'))
ax2.text(((dot2_x_position+device_length/2)/device_length),
↪   np.max(dot2_psix0)-0.05, f"{dot2_x_position*1e9:.1f}nm",
         transform=ax1.transAxes,
         fontsize=10,
         zorder=10,
         verticalalignment='center',
         horizontalalignment='center',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white',
          ↪   alpha=0.8, edgecolor='green'))
```

Examples

The images 6.2 and 6.3 shows, respectively, the confinement results images of the devices `sqd_soi_planar_dp` and `dqd_soi_planar_dp`. Full results are presented in Chapter 8.

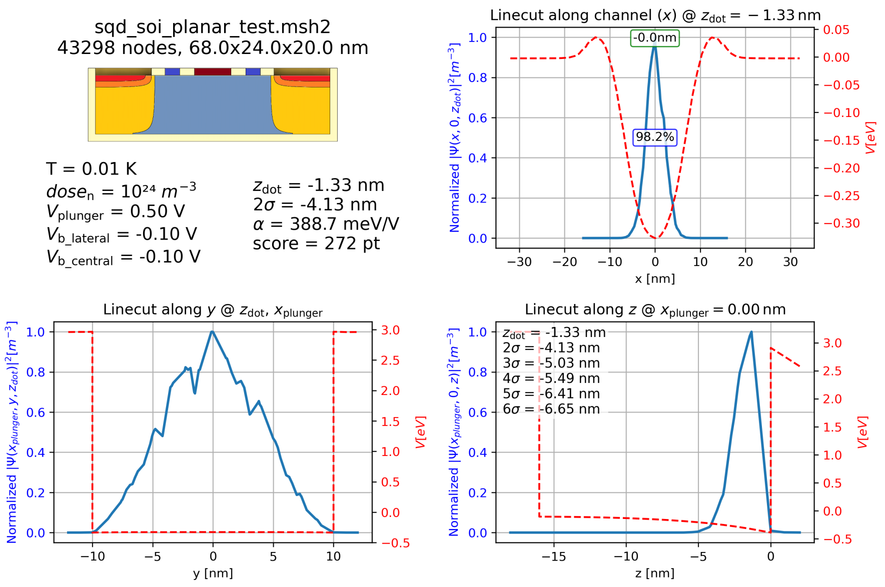


Figure 6.2: `sqd_soi_planar_dp` confinement results image, low mesh node density

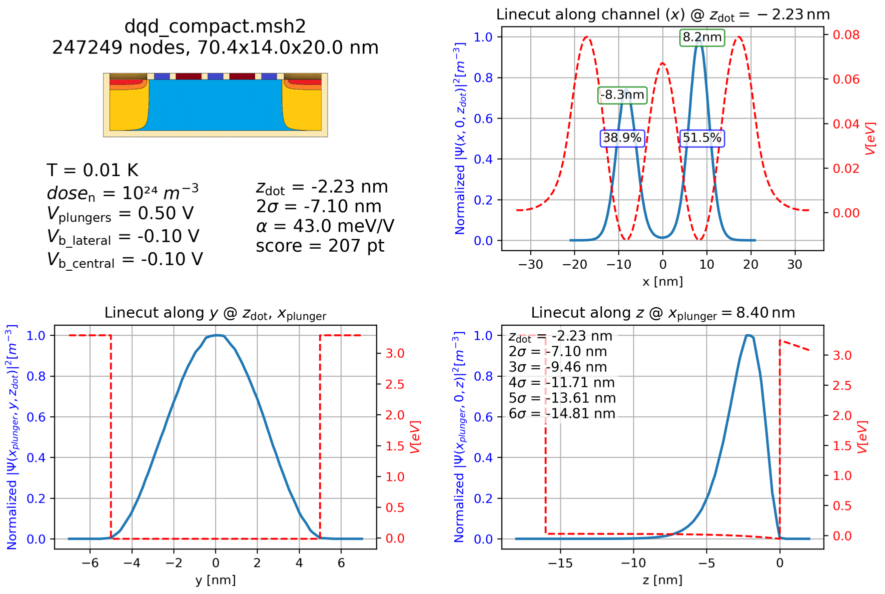


Figure 6.3: `dqd_soi_planar_dp` confinement results image, high mesh node density

6.1.6 Results Analysis in ParaView

ParaView is an open-source software designed for visualizing large and complex datasets. It allows users to explore and analyze data in three dimensions, providing a wide range of tools for rendering, filtering and plotting results. In this work, as suggested by Nanoacademic Technologies, ParaView is used to inspect simulation outputs, giving a clear and intuitive representation of the device physical quantities, which helps in understanding the behaviour of the system.
In this example the analysis of the VTU file containing information about the eigenstate probability density function, defined as $|\psi(x, y, z)|^2$, is shown. The firsts step is to open the file, by using the top menu `File > Open`. Once the file is loaded it is necessary to click the `Apply` to show the mesh in the 3D workspace.

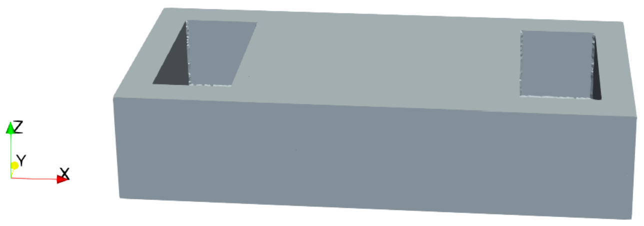


Figure 6.4: ParaView VTU import

The mesh is coloured with respect to the exported value. Right now the mesh is fully visible so the inner colouring is hidden. To show the dots one can set the opacity factor to a lower value than 1, like 0.1, and see the dots by rotating the mesh. Another way is to perform a clip, dividing the mesh by using a tool plane. To do so, the feature "Clip with an implicit function" is used. By modifying the "Origin" and "Normal" values of the plane it can be rotated. By setting the second value, so the y axis, of both fields to 1 and the others to 0 the resulting plane will be an XoZ and it can cut the device in half showing the channel along the x axis.

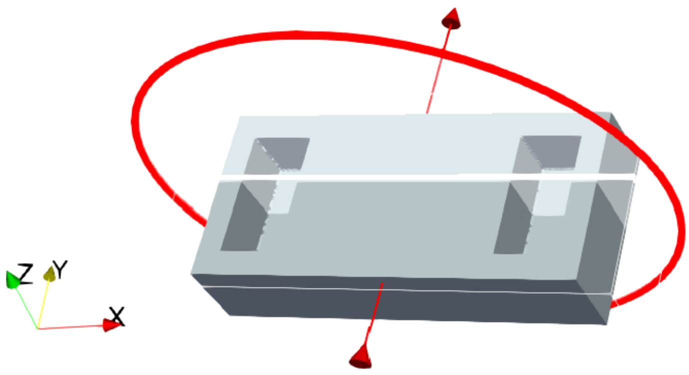


Figure 6.5: ParaView clip command

By deselecting "Show Plane" and "Invert", enabled by default and click `Apply` the clip is complete and the inside of the device became visible. On the right menu it is possible to select a colormap and perform additional operations to the dataset.
An example of how a quantum dot should appear is shown in figures 6.6 for a SQD device and figure 6.7 for a DQD one, where the asymmetry of the two dots is noticeable by the different colouring.

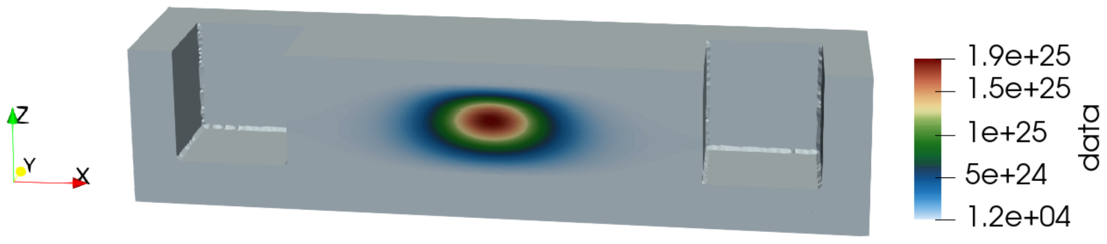


Figure 6.6: Device `sqd_soi_planar` results in ParaView

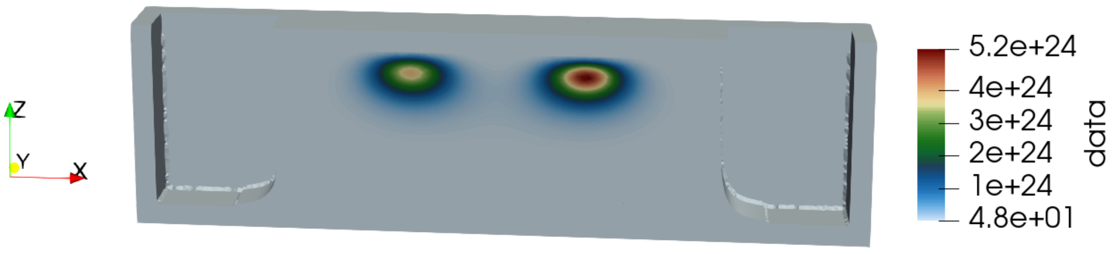


Figure 6.7: Device `dqd_soi_planar_dp` results in ParaView

A line can be defined from two points in space and plotted on a Cartesian graph using the "Plot Over Line" feature. However, it is recommended to use the `device_results.mat` output file from the simulation for processing in Matlab, which allows for more advanced analysis, while in ParaView it is only possible to plot the data and export it as a CSV file.

ParaView is a powerful tool for visualizing and inspecting scientific data, but in this work it was only employed to graphically inspect the quantum dots.

6.2 Batch Runner

The long simulation times needed to understand the effects of a small change in simulation parameters lead to the realization of a pilot python script that, given a set of input parameters, creates a set of configurations and feed those to a series of simulation scripts to be run in parallel as subprocesses. This was called *batch runner* and enables process-level parallelization for the simulations. With this approach, the time required for a single simulation still remains the same, but in that time the runner return multiple results from multiple configurations, for the same device. The pilot code, available in the attachments appendix as script C.1, is now examined in detail.

First of all, it is necessary to understand how many subprocesses can be instantiated while maintaining the machine load under a reasonable threshold. On the server used for the simulations, setting the number of workers (parallel threads) to half of the total CPU count proved to be a good compromise between the number of simulations run simultaneously and the overall system load. To check for that, the `top` command displays real-time information about system resource usage, including CPU and memory consumption, active processes and system load. The latter is shown in the top-right corner under the name `load average`. The first number shows the average over the last minute and should not exceed, as a rule of thumb, two thirds of the available cores. Keeping the load below this threshold ensures system stability by leaving resources available for system processes, I/O and memory management, preventing slowdowns or simulation crashes.

```
WORKERS = int(os.cpu_count()/2)
```

Then, it is possible to define the variables to sweep and define them via a `linspace` command, or manually. Then, the `itertools` module generates combinations of values using those defined earlier. If n variables are defined, each with k values, the total number of combinations will be k^n.

```
# DEFINE PARAMETERS TO WEEP
plunger_values = [0.5, 0.6, 0.8, 1.0, 1.1]
doping_values = [1e18*1e6,1e17*1e6,1e18*1e6,1e19*1e6,1e20*1e6]

# COMBINATIONS
combinations = list(itertools.product(plunger_values, doping_values))
```

As the variables are passed to the simulation script as environment variables, a copy, called `env`, of the current ones is created, in order to append the needed configuration. Then, the base output folder is set. If the `batch_results` folder does not exists, the script creates it.

```
# DEFINE ENVIRONMENTAL VARIABLES
env = os.environ.copy()

# BASE OUTPUT FOLDER
base_output = Path("batch_results")
base_output.mkdir(exist_ok=True)
```

The `run_simulation` function mainly does three things:

- It creates a string with the parameters value and impose that as the current working directory seen by the subprocess. It does this to have inside the `batch_results` folder a set of subfolders named as the simulation configuration. In the next section, the modification added to the simulation script shows that the paths in case of a batch run are modified in order to store the results in the cwd instead of the `results` folder.
- It adds new environment variables to pass the simulation inputs to the subprocess. A boolean flag called `BATCH` was added to let the script know that it was run from the batch runner, modifying some paths and taking the parameters from the environment variables instead of the default ones hard coded inside the script.
- It starts a subprocess calling the script name, in this example `sim_v4.py`, inhibiting its output on the console and passing it the cwd and environment variables.

```
def run_simulation(plunger, doping):
    # Define directory name for the single simulation
    label = f"V{plunger}_N{int(doping):.0e}"
    output_dir = base_output / label
    output_dir.mkdir(exist_ok=True)

    env["V_PLUNGER"] = str(plunger)
    env["N_DOPING"] = str(doping)

    env["BATCH"] = str("True")

    print(f"[INFO] Running sim: {label}")

    subprocess.run(
        ["python", str(Path(__file__).parent/"sim_v4.py")],
        cwd=output_dir,  # save files inside simulation directory
        env=env,
        stdout=subprocess.DEVNULL
    )
```

Finally, when the script is run from the terminal, the `ProcessPoolExecutor` instantiates the predefined number of workers, each handling one of the required combinations. If the number of combinations exceeds the number of workers, new subprocesses can only start once a previously running thread has completed its execution.

```python
# PARALLELIZZAZION
if __name__ == "__main__":
    with ProcessPoolExecutor(max_workers=WORKERS) as executor:
        futures = [executor.submit(run_simulation, v, n) for v, n in
         ↪  combinations]
        for f in futures:
            f.result()

merge_dot_results()
```

At the end, the custom function `merge_dot_result` merges all the desired output variables, stored in the file `device_results.mat` for each configuration, into one Matlab file. This final file is then used to perform the analysis across all configurations and observe correlation between parameters. The function is available in the attachments appendix inside the script A.4.
An example of merged result, based on a batch simulation, is observable in figure 6.8, where the wavefunction peak location in the z axis and relative cumulative confinement levels starting from the silicon interface with the gate oxide, are put in relation with the applied plunger voltage and source/drain doping concentration.

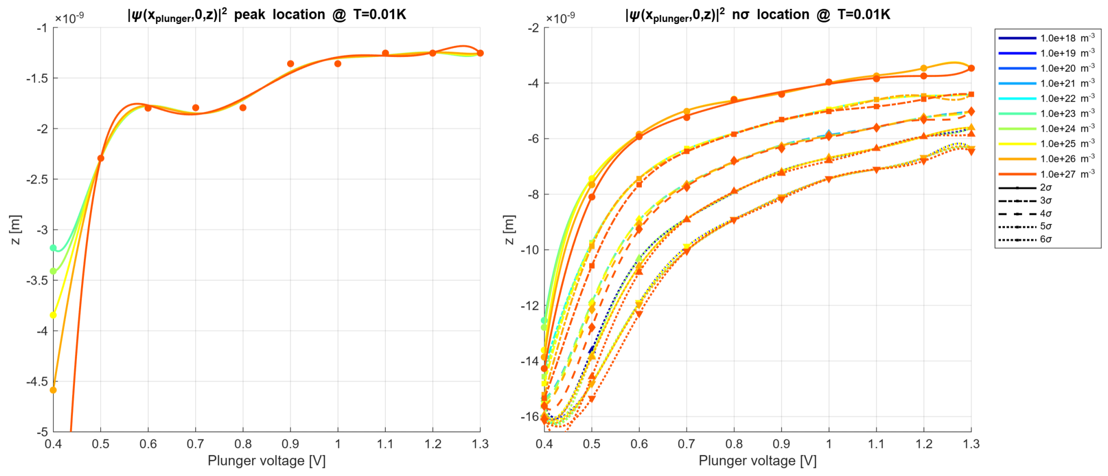


Figure 6.8: Example of batch run result

In this case, the plot was generated in Matlab with a dataset of 600 values from a batch of 100 simulations, requiring about 10 minutes only.

Modifications To The Simulation Script

To perform a batch run, the simulation script needs to be modified in order to work in both conditions of a single or batch run. Mainly, the script need to know if it is in a batch run to update some paths and retrieve the configuration from the environment variables. The values from the environmental variables can be retrieved with the command `os.getenv`, that take as argument a variable tag and a default variable to adopt if the addressed one does not exist. The first one, is the `BATCH`, that tells the program if it was run from a batch. The default value `False` is used to not adopt these modifications in a single simulation. If the variable is true, thus existing inside the environment variables, this means that the cwd of the script is changed as `simulation/batch_results/config_name` where `config_name` can be something like `V0.5_N1e+25`. Modification to the paths of the input files are done at the beginning of the script in order to still retrieve the files from the `simulation/config` directory and to save HDF5 and Matlab files inside the cwd.

```
# IMMEDIATELY FIND OUT IF BATCH RUNNING IS ENABLED
batch = os.getenv("BATCH", "False")

# CHECK IF WE ARE IN BATCHING AND UPDATE SOME PATHS
if batch == "True":
  path_parameters = Path("../../config/exported_parameters") /
  ↪  parameters_name
  path_hdf5 = "device_results.hdf5"
  path_results_mat = "device_results.mat"
```

Then, a default value for the variables used for simulation can be defined, used when the environmental tag does not exist, thus indicating the script was run singly. Again, the values are retrieved with the `os.getenv` command.

```
default_n_doping = 1e18*1e6
default_plunger_voltage = 0.5
n_doping    = float(os.getenv("N_DOPING",  default_n_doping))
V_plunger_1 = float(os.getenv("V_PLUNGER", default_plunger_voltage))
```

Lastly, if we are in a batch run the image need to be saved directly inside the cwd the runner defined, in order to avoid excessive use of subfolders. The `text_loc` variable is explained in the next section.

```
# SAVE IMAGE
if batch == "True":
  path_results_img = "../" + text_loc + "_" +
  ↪  f"V{V_plunger_1}_N{int(n_doping):.0e}" + ".png"
plt.savefig(path_results_img, dpi=dpi, bbox_inches='tight')
```

6.2.1 Scoring System

The batch runner allows the simulation of multiple configurations in parallel, thus saving time, with a speed-up factor proportional to the number of instantiated workers. Having a large number of results introduces the issue that analysing them requires considerable time, which counteracts the benefits and time savings achieved through batch execution. To address this problem, a scoring system is introduced. The working principle is to assign a score to the configuration. The simulation adds points if some criteria have been met and removed if other problems occurs. Firstly a `score` variable needs to be defined at the beginning of the code.

```
score = 0
```

Then, after Poisson and Schrödinger solvers are done, it is possible to assign score point with the processed results. An example for a double quantum dot device could be:

```
# UPDATE DEVICE CONFIGURATION SCORE
score += (dot1_x_confinement/0.5)*100
score += (dot2_x_confinement/0.5)*100
score += 50*np.exp(-5*np.abs(np.abs(dot1_x_position*1e9) -
 ↪  np.abs(plunger_x_coordinate*1e9)))
score += 50*np.exp(-5*np.abs(np.abs(dot2_x_position*1e9) -
 ↪  np.abs(plunger_x_coordinate*1e9)))
score += 100*np.exp(-np.abs(dot_z_position*1e9))
score -= 100*np.exp(-5*np.abs((dot1_x_confinement/0.5) -
 ↪  (dot2_x_confinement/0.5)))
```

The example code attempts to assess the confinement of individual dots as well as the interdot coupling and asymmetry. Unfortunately, it currently struggles to provide an accurate assessment of system performance. Fine-tuning of the scoring algorithm is required to achieve the desired robustness of the paradigm.
Then the score can be displayed in the output image inside the text in the first quadrant, as shown in Section 6.1.5. The variable `text_loc` is set as the prefix to the file name. It is set to an X if the dot is not localized under the plunger gate, therefore identifying a not-working configuration. They can be automatically deleted by the use of the script `remove_fails.py`, available in the attachments appendix as script A.9. Otherwise, it is set to the score.

```
text_loc="X"
if localized_dot:
  text_loc=f"{int(score)}"
```

By ordering alphabetically the output images, which are all stored in the `batch_results` folder, it becomes possible to instantly identify the best-performing configurations.

6.3 Transport Layer Simulation

Quantum transport simulations offer a way to study how carriers move through a device and how it responds to different electrical conditions. By explicitly modelling the flow of carriers, the transport simulations provide detailed insight into the mechanisms that govern device operation, including tunnelling, confinement effects and blockade phenomena. They enable the evaluation of device performance from a fully quantum mechanical perspective, capturing effects that classical transport models cannot describe. This approach allows for predicting the behaviour of nanoscale devices, assessing their stability and identifying the factors that influence their electronic properties, making it an essential step in the design and optimization of quantum systems.

QTCAD provides a framework for simulating quantum transport, employing a many-body approach, including Coulomb interactions between electrons. Coupling between the device and the leads is modelled through tunnelling events, described by a Hamiltonian based on single-electron eigenfunctions. The Hamiltonian is defined in the so called Fock basis, a way to describe a quantum system in terms of the number of particles occupying each possible state. Each basis state specifies how many particles are in each level, making it ideal for systems with indistinguishable particles like electrons or photons. It naturally accounts for quantum statistics, such as the Pauli exclusion principle for fermions and is widely used to model many-body phenomena like single-electron tunnelling and Coulomb blockade.
This approach enables the simulation of transport phenomena under strong quantum confinement, capturing effects such as Coulomb blockade. The junction concept in QTCAD allows for the computation of Coulomb peaks and charge-stability diagrams, which are crucial for analysing the single-electron regime in quantum devices. [47][45]
It also supports transport simulations via the non-equilibrium Green's function (NEGF) formalism, which accounts for non-equilibrium quantum statistics and can model both classical transport, like thermionic emission and quantum transport, including tunnelling through potential barriers. The NEGF module enables the calculation of charge density and electric potential under nonequilibrium conditions through NEGF-Poisson simulations and allows for the evaluation of electric current, density of states and other relevant quantities. However, the NEGF approach does not include electron-electron interactions, so it cannot capture Coulomb blockade effects. [44] For simulations where Coulomb interactions are important, the junction module remains the preferred choice.

In this work, the transport layer was employed to simulate Coulomb interactions, enabling the generation of the charge stability diagram. This diagram provides insight into the blockade regimes, the number of carriers confined within a specific dot in single quantum dot devices and the particle addition spectrum of double quantum dots.

6.3.1 Lever Arm

Although the lever arm is an intrinsic property of the device, it is discussed in the transport section because of its greater relevance in this context rather than in confinement. The term lever arm (or leverarm) α_G refers to the proportionality factor that links variations in gate voltage φ^G_{bias} to changes in the electrochemical potential μ of the dot. [43]

$$\mu = \mu_0 - e \cdot \alpha_G \cdot \varphi^G_{bias} \tag{6.1}$$

where e is the elementary charger and $\mu_0 = \mu|_{\varphi^G_{bias}=0}$. The electrochemical potential for a transition between the (N-1)-electron and N-electron ground states of a quantum dot is

$$\mu(N) = E_{tot}(N) - E_{tot}(N-1) \tag{6.2}$$

where E_{tot} is the total energy of the dot in the N-electron ground state. [43]
Within the constant-interaction model, the lever arm of gate G on the dot is given by the ratio of the capacitance between the dot gate G and the self capacitance of the dot.

$$\alpha_G = -\frac{C_{0G}}{C_{\sum}} \tag{6.3}$$

In QTCAD, the lever arm can be determined without explicitly evaluating the individual capacitances, since the full device geometry is taken into account rather than approximating it with a lumped-element circuit model. The approach relies on directly computing how the electronic structure of the quantum dot responds to variations in gate bias, from which the lever arm is obtained through a linear fit of the resulting behaviour. [43]

To handle the lever arm inside the simulations many scripts are involved. Mainly, the script `get_leverarm.py` manages the coefficients inside a Matlab file called `lever_arm.mat`, which are created by the script `compute_leverarm.py`. Both are available in the attachments appendix as, respectively, script A.11 and A.10. The paradigm is to obtain the lever arm coefficient for a device gate in a straightforward manner, without having to compute it each time, since such simulations are often time-consuming. It if was computed previously, it simply retrieve the coefficient from the `.mat` file, otherwise it calculates it. It is important to have the Matlab file before starting a batch run, otherwise many instances are created, each one trying to write to the same file, leading to errors. The main functions used to handle the coefficients are now explained.

```python
def get_lever_arm(device, boundary):
    mat_file_path = get_parent_mat_path()
    if not os.path.isfile(mat_file_path):
        compute_lever_arm(device, boundary)

    mat_contents = sio.loadmat(mat_file_path)

    lever_arm_value = float(mat_contents['lever_arm'].squeeze())
    return lever_arm_value
```

This function can be called from the simulation script as

`leverarm = get_lever_arm(d, "gatebnd_plunger_1")`

and checks for the presence of the lever arm Matlab file, otherwise it calls the function `compute_lever_arm`, now analyzed.

Each lever arm computation, whether for a single or multiple values, is carried out by defining the parameters for the Poisson and Schrödinger solvers, which are included within the lever arm calculation. To speed up the simulation, the number of states to be simulated can be reduced.

```
params_poisson = PoissonSolverParams()
params_poisson.tol = 1e-5
params_schrod = SchrodingerSolverParams()
params_schrod.num_states = 10

lever_arm_solver_params = LeverArmSolverParams({
    "pot_solver_params": params_poisson,
    "schrod_solver_params": params_schrod
})
```

Then, it is possible to choose the span of the gate voltages. If a single coefficient is needed, it is sufficient to specify a bias value and then perform simulation for that value and for two additional ones at a ϵ distance, in the example set to $0.05V$. If the whole curve of the proportionality factor is needed, a `linspace` command is used to define a range of gate voltage values, in the example from $0V$ to $1V$ with 21 values, corresponding to a $0.5V$ step size.

```
# Example of a reduced voltage set
plunger_voltages = [bias-0.05, bias, bias+0.05]

# Example of a voltage set defined with linspace
plunger_voltages = np.linspace(0, 1, 21)
```

Then, the lever arm solver can be instantiated by passing the device, previously defined in the simulation script and passed as argument by the intermediary script `get_lever_arm.py`, the list (in this case just one value) of surface boundary labels, the voltages, the dot region and finally the parameters. By calling the `solve` function, the lever arm calculation is initiated and for each voltage value, a simulation is performed, therefore requiring a significant amount of time to complete. Then, the lever arm value can be returned by taking the absolute value of the first coefficient (ground state) and dividing it by the elementary charge to have it in eV/V.

```
lever_arm_slv = LeverArmSolver(device,
                               [boundary],
                               plunger_voltages,
                               dot_region=device.dot_region,
                               solver_params=lever_arm_solver_params)
poly_coeffs = lever_arm_slv.solve()

lever_arm_value = np.abs(poly_coeffs[0]) / ct.e
```

Then, for each simulated energy state, the results are stored inside the Matlab file as the pair `bias:energy`, as well as the ground state lever arm coefficient.

```
mat_data = {"lever_arm": lever_arm_value}

for i, data in enumerate(lever_arm_slv.energies.T / ct.e):
    ax1.plot(plunger_voltages, data, label=f"state {i}")
    # Save bias and energy for each state
    mat_data[f"bias_{i}"] = plunger_voltages
    mat_data[f"energy_{i}"] = data

# Save the dictionary into a .mat file
savemat("lever_arm.mat", mat_data)
```

In figure 6.9 it is possible to observe the lever arm curves for different energy states related to a MOS-like SOI planar device. In Chapter 8, the main parameters affecting the lever arm coefficients are reviewed.

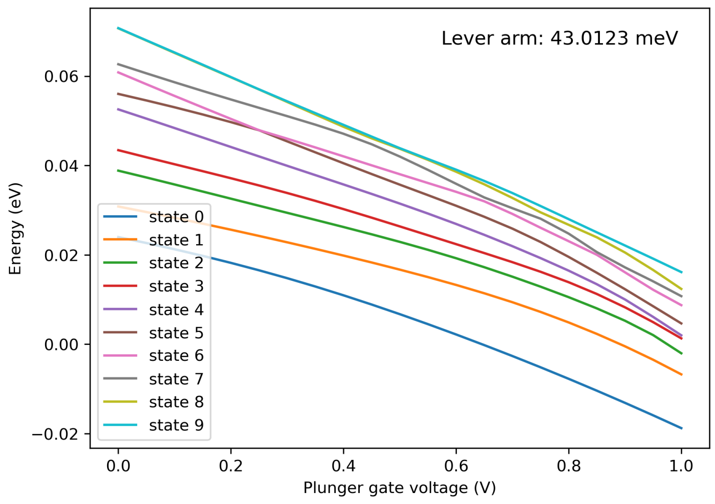


Figure 6.9: Example of lever arm computation

6.3.2 Coulomb Diamonds

The term Coulomb diamonds refers to the diamond-shaped regions that appear in a charge stability diagram of a quantum dot. They indicate Coulomb blockade, revealing the discrete electron occupation states and the energy required to add or remove electrons from the dot. By analysing these diamonds, it is possible to know the number of confined carriers of a quantum dot. To compute the diagram, a complete Poisson-Schrödinger simulation need to be carried out. After that, it is possible to call the function `compute_charge_stability_diagram` that takes as argument the device to simulate and automatically computes the diagram. After the full device package simulation, the many-body solver is instantiated, a package designed to include electron-electron interactions that cannot be captured within a single-particle approximation. By solving the many-body Hamiltonian, it provides access to correlated electronic states, charge configurations and excitation spectra in quantum dots. Here the parameters can be set in order to take into account the levels to keep and the lever arm coefficient. Then, a junction is defined, representing a transport system consisting of a quantum dot connected between two leads (source and drain). It enables simulation of electron transport through sequential tunnelling. By using these two object it is possible to compute the charge stability diagram. Unlike the code developed for the device simulation using the `device` package, the implementation for the transport simulation closely follows the original QTCAD tutorial. For this reason, the full code will not be analyzed in detail within this work. Instead, it can be accessed directly at the tutorial official webpage:
`https://docs.nanoacademic.com/qtcad/practical_application/6-stability/`
The full code, including the modifications required to adapt it to these simulations, is available in the attachments appendix as script B.2. As an example, the charge stability diagram of the SQD device `sqd_soi_planar_dp` is shown in figure 6.10.

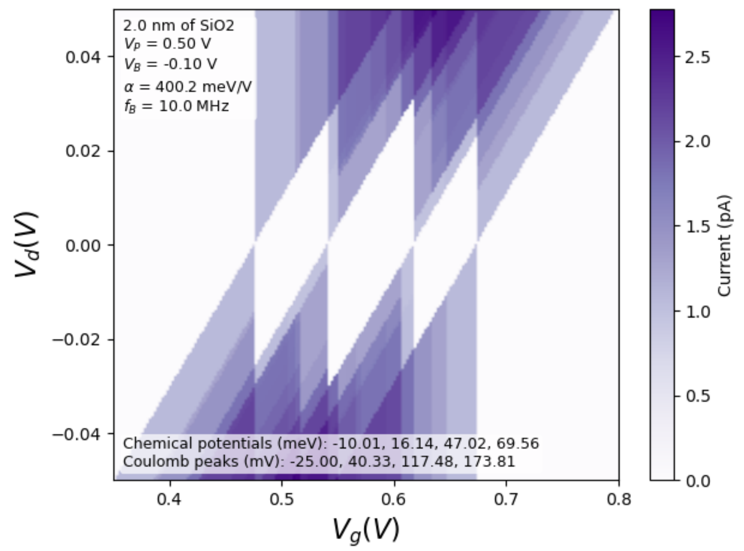


Figure 6.10: Coulomb diamonds of the device `sqd_soi_planar_dp`

6.3.3 Particle Addition Spectrum

A charge stability diagram for a double quantum dot takes the name "particle addition spectrum". It is a graphical representation of the electron occupancy in each dot as a function of the applied gate voltages, showing regions of stable charge configurations separated by lines where electron transitions occur due to Coulomb blockade effects. The diagram provides insight into interdot coupling, charging energies and the electronic configuration of the system, making it a key tool to characterize and control double quantum dots. The goal is to analyze electron occupancy and Coulomb blockade regions in a double quantum dot device. A full simulation would sequentially solve the Poisson, Schrödinger, many-body and master equations for each gate bias, but this is computationally intensive. To reduce cost, approximations such as the lever arm method, Hubbard model simplifications and near-equilibrium response functions are used. These approaches allow estimation of energy level shifts, Coulomb interactions and charge transitions efficiently. As for the Coulomb diamonds computation, the implementation for this diagram closely follows the original QTCAD tutorial. For this reason, the full code will not be analyzed in detail within this work. Instead, it can be accessed directly at the tutorial official webpage: `https://docs.nanoacademic.com/qtcad/tutorials/transport/double_dot_stability/` The full code, including the modifications required to adapt it to these simulations, is available in the attachments appendix as script B.3. As an example, the charge stability diagram of the DQD device `qqd_soi_planar_dp`, with the addition of dot occupancies, is shown in figure 6.11.

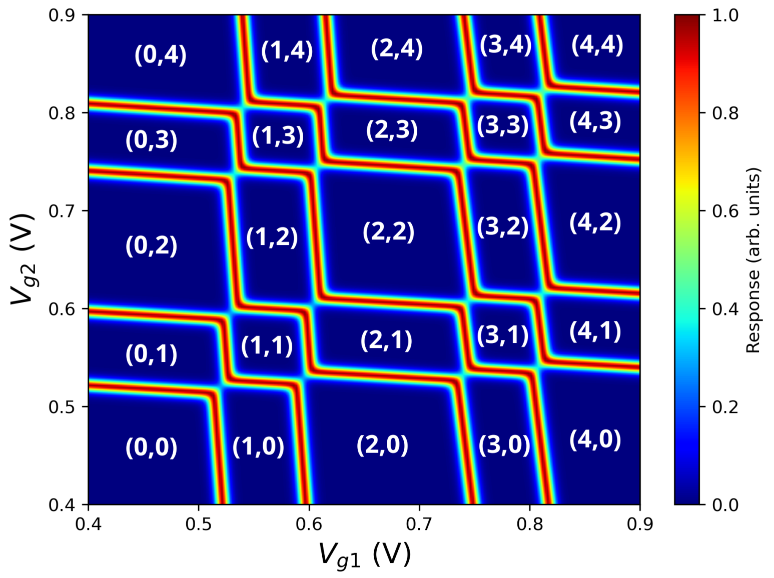


Figure 6.11: Particle addition spectrum with dot occupancies of the device `dqd_soi_planar_dp`

Chapter 7

Fabrication Process Simulation

7.1 Engineer A Wafer For Quantum Computing

As reviewed in Section 3.1, the primary material for carrier confinement is isotopically purified (or isotopically enriched) ^{28}Si, the isotope of silicon which has 14 neutrons inside the nucleus. Besides the fact that it is the most abundant isotope in nature, it has zero nuclear spin, not interacting with the carrier spin and so solving the problem of hyperfine interaction. To host spin qubits it is necessary to use a high purity (3N or above) silicon alloy that unfortunately it is not present on state-of-the-art ready to sell stocks of wafers. It is so compulsory to grow a layer of ^{28}Si before any other fabrication step. Veldhorst et al. demonstrated that using isotopically enriched silicon can increase decoherence times, with T_2 extended by up to 30x and T_2^* by up to 120x. [52]
The two main growth techniques are:

- Epitaxial growth of ^{28}Si on a seed layer: here it is possible to use a FD-SOI wafer with the thinnest silicon layer possible, both by buying it thin or by doing a partial etchback of the silicon overlay. The latter is not recommended because, even when highly diluting the acids used for silicon etching, extreme caution is required. At nanometer scale even a slight over-etch can remove a significant amount of material.
- Deposition of ^{28}Si on oxide layer: this method involves complex processes such as hybrid MBE/CVD, ALD, or ALE (Atomic Layer Epitaxy) and therefore was not analyzed in detail.

Due to the complexity of the process, the isotopically enriched silicon layer was assumed to be grown epitaxially, starting from a silicon seed located in the overlay of a pre-fabricated Silicon-On-Insulator wafer. In the simulations, the silicon channel thickness results from the combination of the original overlay and the epitaxially grown enriched layer. By analyzing the z-axis cumulative confinement levels, it is possible to assume how hyperfine interactions would influence both the spatial confinement and the overall qubit behaviour. It is therefore essential to maximize the distance between the expected quantum dot location and the non-enriched silicon layer. An in-depth analysis of hyperfine interactions was not addressed, since it goes beyond the objectives of this work.

7.1.1 Prefabricated Wafer Selection

In order to have a state-of-the-art substrate for the analyzed devices, the wafer was selected to be readily available for purchase from a European manufacturer. Some research was conducted to identify a supplier for a quantum-ready wafer available in reasonable quantities. Unfortunately, no retailer was found that could provide the required specifications. Therefore, the device was designed using planar SOI technology, given the widespread availability of SOI wafers, which have been well established and widely used in the industry for several decades. One of the largest and well known manufacturer of wafer in Europe is Soitec, a French company and a global leader in semiconductor materials, specialized in the production of SOI wafers, with manufacturing sites in France, Singapore and China. Its wafers are used in a wide range of applications, including mobile devices, automotive components, optical sensors and radio-frequency circuits. For the following section, where the main fabrication steps are done in order to simulate the manufacturing of the device, a SOI wafer from Soitec FD-SOI wafer line was chosen. In addition to the typical benefits of this type of wafer for standard CMOS technology, it was chosen for having the thinnest silicon overlay, serving as a seed layer for the isotopically enriched layer and maximizing the enriched silicon over the standard alloy thickness ratio. A wafer with a $6nm$ overlay was chosen, onto which an additional $10 \div 15nm$ of ^{28}Si needs to be grown. The thickness of the enriched silicon layer can be increased if the z-axis confinement indicates that a quantum dot is too close to the standard alloy, potentially being affected by hyperfine interactions.
For simulations, the buried oxide thickness is not critical, as it can be reduced to save simulation time and generate a lighter mesh file. It only becomes relevant when a backgate is used, in which case the actual thickness must be considered to accurately compute the capacitive coupling.

7.1.2 Epitaxial Growth of Isotopically Enriched Silicon

To grow a silicon layer it is necessary to use silane gas (SiH_4). Few companies in Europe manufacture the enriched silane gas for small volume manufacturing. One of the companies is Orano Group and its activity called Stable Isotopes. Operating from its specialized laboratory at *Centre Nucléaire de Production d'Électricite du Tricastin* (southern France), Orano applies advanced centrifugation technology, originally developed for uranium, to purify and enrich isotopes such as silicon-28, xenon-136 and molybdenum. They produce enriched ^{28}Si under the chemical form of SiF_4, with an isotopic grade over 4N (99.99%) and deliver under gaseous form ($^{28}SiF_4$ and $^{28}SiH_4$) or silica ($^{28}SiO_2$). Knowing that there is a readily accessible manufacturer of isotopically enriched silane in Europe provides an excellent starting point, supporting the feasibility of production at PiQuET and confirming the theoretical framework.

7.2 Capabilities and Limitations at PiQuET

PiQuET (Piemonte Quantum Enabling Technology) is a state-of-the-art applied research infrastructure located in Turin, Italy. Established under the POR FESR 2014-2020 initiative and co-managed by the National Institute of Metrological Research (INRiM), Politecnico di Torino and University of Turin, its mission is to enable advanced research and industrial innovation in quantum, micro and nanotechnologies. It features a $400m^2$ ISO-classified cleanroom, specialized laboratories in quantum metrology, microfluidics and additive manufacturing. PiQuET supports prototyping, device characterization, rapid material development and customized fabrication services.
PiQuET's clean-room facilities provide a wide range of advanced micro- and nanofabrication, as well as characterization tools. In particular, they support:

- Quantum device characterization (quantum clocks, atomic sensors, quantum electronic, photonics) for metrology and standard development.
- Deposition, Growth and Thermal Treatment (ALD, LPCVD, ICP-CVD, sputtering, graphene CVD, hot embossing, thermal processing).
- Lithography across diverse methods including laser writing, UV exposure, mask alignment, 2PP and upcoming EBL.
- Dry and wet etching, with secure wet processing environments.
- Chemical processing, from cleaning and resist handling to CMP and metal plating under clean conditions.
- Device packaging technologies such as flip-chip, wire bonding, wafer bonding, dicing and embossing.
- Advanced characterization instrumentation (SEM, FIB, TEM, AFM, probe stations, ellipsometry, profilometry) for assessing micro and nanodevices.

Throughout the development of this work, the possibility of fabricating a potential device at PiQuET has always been taken into account, given its close collaboration with the Politecnico di Torino. This perspective ensured that the proposed concepts were considered not only from a theoretical standpoint but also with regard to their practical realization in a state-of-the-art cleanroom environment. However, the current absence of some key equipment, such as the ion implanter and the electron-beam lithography system (still in the installation phase), has led not only to the analysis of state-of-the-art fabrication processes but also to the investigation of possible alternatives aimed at adapting the process flow and making the realization of a device at PiQuET feasible.
Since for small production volumes mask-based lithography is not usually employed due to the high costs, it is mandatory to rely on electron-beam lithography, meaning that fabrication must wait until this tool becomes available. For the device doping, thermal diffusion and silicide processes have been considered as alternatives to ion implantation.

7.3 Device Fabrication Process in Sentaurus

After confirming that a device exhibits proper confinement behavior through simulations in QTCAD, it is fundamentally important to simulate its fabrication in a TCAD tool such as Synopsys Sentaurus. The aim of this work is to adapt state-of-the-art VLSI and CMOS processes to leverage existing manufacturing facilities for the realization of quantum devices based on gate-defined quantum dots on silicon substrates. The process simulation work was carried out together with Leonardo Ossino, an intern at Politecnico di Torino, who participated to study semiconductor qubit devices and to complement his training by using a professional TCAD tool such as Synopsys Sentaurus. His work was supervised by the candidate and Nicola Carbonetta, a PhD student who also supervised the candidate's thesis work and provided guidance throughout the project.

Although many device variants were simulated in Sentaurus, the main focus was a planar SOI double quantum dot. The study of a DQD device was conducted at the end of the work, thereby building on the analysis of solid-state devices and quantum confinement, as well as the investigation of interdot coupling and phenomena related to multidot configurations. For the device, referred as `dqd_soi_planar_dp`, the geometry and dimensions are shown in figure 7.1. This model incorporates the doping profiles derived from the simulation of an identical structure in which those regions were not modelled.

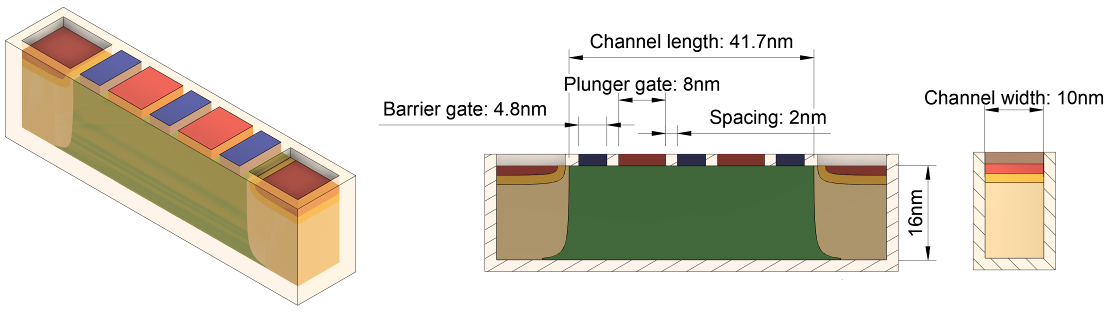


Figure 7.1: Device to be simulated in Sentaurus

As a starting point, the wafer, described in Section 7.1.1, was selected as a generic Soitec FD-SOI with a $6nm$ silicon overlay. Next, as reviewed in Section 7.1.2, the epitaxial growth of isotopically enriched silicon via silane gas was carried out. Shallow trench isolation is then performed to isolate the device as a single silicon island. The source and drain doping was performed via thermal diffusion, since the ion implanter is not yet available at PiQuET. After the deposition of the gate oxide ad the patterning of the metal gates, the geometry is mirrored to produce the final device.

The main simulation steps are now described.

First, certain dimensions are defined to specify the substrate and model only a quarter of the actual device, reducing computational time. By defining the three main layers, the SOI wafer substrate is defined. The use of tags makes coordinate values accessible throughout the entire code, while the use of variables increases its reusability. The full geometry can later be reconstructed using two mirroring steps.

```
# ------------------------------------------------------------- #
# STEP 1: SILICON OVER INSULATION WAFER DEFINITION              #
# ------------------------------------------------------------- #

line x location= 0.0<nm>        tag= Si28_top
line x location= @t_Si28@       tag= Si_ov_top
line x location= 16.0<nm>       tag= BOX_top
line x location= 36.0<nm>       tag= substrate_top
line x location= 86.0<nm>       tag= substrate_bottom

line y location= 0.0            tag= Mid_ch
line y location= @L_halfCh@     tag= End_ch
line y location= 33.2<nm>       tag= End_drain
line y location= 43.2<nm>       tag= Right_STI
line y location= @L_halfWafer@  tag= End_wafer

line z location= 0.0            tag= Back_ch
line z location= @W_halfCh@     tag= Front_ch
line z location= 15.0<nm>       tag= Front_STI
line z location= @W_halfWafer@  tag= Front_wafer

# Silicon overlay
region Silicon  xlo= Si_ov_top xhi= BOX_top   ylo= Mid_ch yhi=
↪  End_wafer zlo= Back_ch zhi= Front_wafer

# Buried oxide
region Oxide xlo= BOX_top   xhi= substrate_top  ylo= Mid_ch  yhi=
↪  End_wafer  zlo= Back_ch zhi= Front_wafer

# Silicon substrate
region Silicon  xlo= substrate_top xhi= substrate_bottom ylo= Mid_ch
↪  yhi= End_wafer zlo= Back_ch zhi= Front_wafer

init !DelayFullD
```

Next, the enriched silicon layer needs to be grown on top of the silicon overlay of the wafer, used as a seed layer. As explained in Section 7.1.2, the growth of this layer needs to be done epitaxially. To model this process more realistically, an epitaxial ambient was defined, instead of performing a simple deposition. The shape of the growing epitaxial layer can be controlled using lattice kinetic Monte Carlo, which allows for more realistic deposition profiles without the computational cost of a fully atomistic approach. To enable this mode, it is necessary to specify `lkmc` in the diffuse command and set the PDB (Parameter Database Browser) parameter as `KMC Epitaxy`. Then, the epitaxial growth is activated by specifying an Epi-type ambient in the `diffuse` command. During the diffusion step, the same equations applied to single-crystal silicon are solved for the epitaxial layer. The grow rate was set as the desired ^{28}Si layer thickness with an additional value of $2nm$. This was done in order to compensate for the thin film oxide, later realized via dry oxidation, that leads to the consumption of a portion (around 45%) of the silicon underneath.

```
# -------------------------------------------------------------- #
# STEP 2: 28-SI EPITAXIAL GROWTH                                 #
# -------------------------------------------------------------- #

pdbSet KMC Epitaxy 1
diffuse time= 1<s> temperature= 550 Epi lkmc epi.thickness= [expr
 ↪  @t_Si28@+0.002]
```

The final substrate, composed by the original Soitec wafer and the epitaxially grown enriched layer, is shown in figure 7.2.

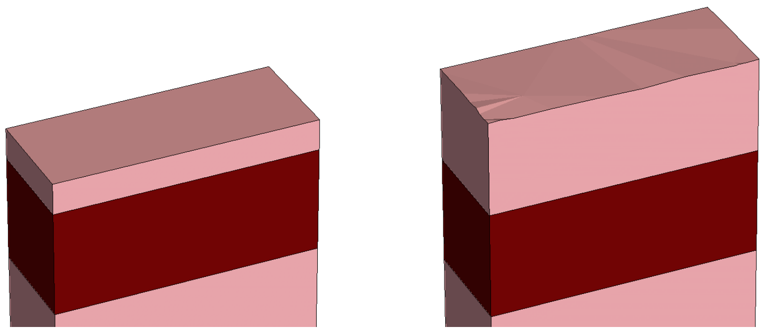


Figure 7.2: Original wafer (left) and epitaxial growth of enriched layer (right)

Next, a shallow trench isolation (STI) is performed in order to isolate the active area of the device, forming an island, to shield it from surrounding silicon. A new mask is defined in order to perform a dry etch via reactive ion etching (RIE). A dry oxidation step follows to create an oxide thin film as a seed to deposit a thick layer via CVD to fill the trench. After a chemical mechanical polishing (CMP) step, the device is planarized and the silicon layer is freshly exposed.

```
# --------------------------------------------------------------- #
# STEP 3: SHALLOW TRENCH ISOLATION                                #
# --------------------------------------------------------------- #

mask name= Si_etching left= -1<nm>   right= 33.2<nm> back= -1<nm>
 ↪  front= 5<nm>
mask name= Si_etching left= 43.2<nm> right= 50.0<nm> back=-1<nm>
 ↪  front= 20.0<nm>
mask name= Si_etching left= -1<nm>   right= 50.0<nm> back=15.0<nm>
 ↪  front= 20.0<nm>

etch material= {Silicon} type= anisotropic time= 1<min> rate= {0.02}
 ↪  mask= Si_etching

diffuse temperature=900<C> time=1<s> O2

deposit material= {Oxide} type= anisotropic time= 1<min> rate= {0.05}

etch material= all type= cmp coord=0.0
```

In figure 7.3, the main steps for STI are shown.

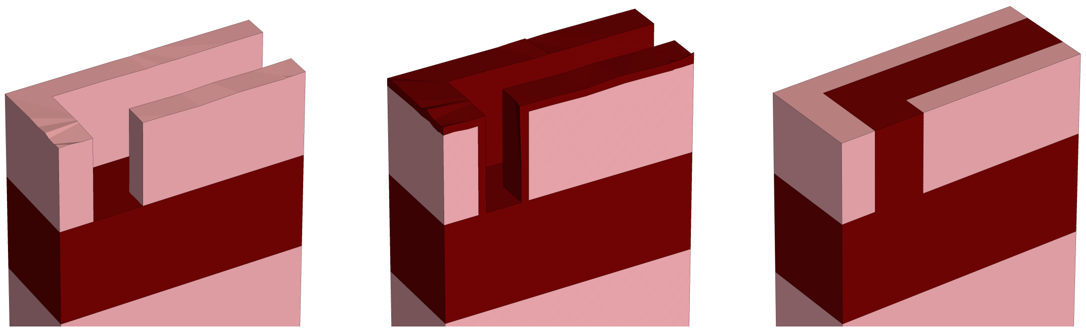


Figure 7.3: Shallow trench isolation steps: RIE, dry odidation, oxide deposition + CMP

Next, it is necessary to mask the device to avoid doping unwanted regions. Before that, a thin nitride layer is deposited. This prevents retrograde doping, ensuring that the dopant profile peak remains near the silicon surface rather than being excessively buried. A thick oxide layer is then deposited to mask the channel, preventing it from incorporating dopant impurities. While depositing the dopant agent, a thermal activation will make it penetrate inside this sacrificial oxide layer instead of the silicon channel. The channel is kept intrinsic to minimize noise arising from crystal imperfections. If doping were introduced, free carriers would be present, which are likely to have poor confinement due to their relatively high energy, even at low temperatures, making it difficult to localize them in a quantum dot and potentially reducing coherence times. Additionally, the dopant-induced electric fields could perturb the confinement potential, introducing local dipoles due to differences in electronegativity and atomic number, thereby increasing the risk of quantum decoherence and shortening qubit lifetimes.

```
# ------------------------------------------------------------ #
# STEP 4: THIN NITRIDE AND THICK OXIDE DEPOSITION FOR MASKING   #
# ------------------------------------------------------------ #

deposit material= {Nitride} type= anisotropic thickness= 1.5<nm>

deposit material= {Oxide} type= anisotropic thickness= 30.0<nm>
mask name= hard_mask left= -1 right= 25.2<nm> back= -1 front=
 ↪  @W_halfWafer@
etch material= {Oxide} type= anisotropic thickness= 30.01<nm> mask=
 ↪  hard_mask
```

In figure 7.3, the two main depositions are shown.

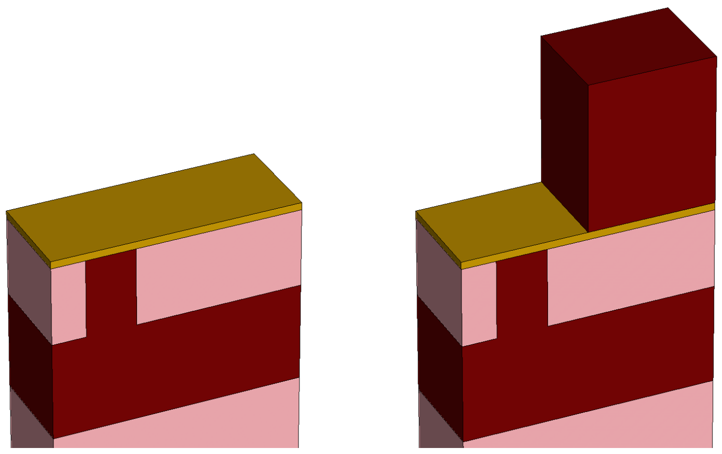


Figure 7.4: Thin nitride and thick oxide masking deposition

Next, the n-type doping is carried out. Before that, a mesh refinement is done in the drain region in order to increase the resolution and better visualize the doping profile. A phosphorus implant is performed to introduce the dopant. Although an ion implanter is not available, the `implant` command was used to simulate the surface doping of the wafer. A subsequent thermal annealing step is carried out to activate the dopant. The hard mask (nitride and oxide), are then removed by two separate etching steps. These steps are shown in figure 7.5.

```
# -------------------------------------------------------------- #
# STEP 5: DOPING VIA THERMAL DIFFUSION                           #
# -------------------------------------------------------------- #

refinebox Silicon min= {0.0 0.0 0.0} max= {0.016 0.0332 0.005}
↪  xrefine= {0.001 0.001 0.001} yrefine= {0.01 0.001 0.001} zrefine=
↪  {0.001 0.001 0.001} add
grid remesh

implant Phosphorus dose= 1.0e+10<cm-2> energy= 1.5<keV> tilt=
↪  45.0<degree> rotation= 0.0<degree>

diffuse temperature= 500<C> time= 0.1<s>

# -------------------------------------------------------------- #
# STEP 6: HARD MASK REMOVAL                                      #
# -------------------------------------------------------------- #

etch material= {Oxide}   type= anisotropic thickness= 30.01<nm>
etch material= {Nitride} type= anisotropic thickness= 1.501<nm>
```

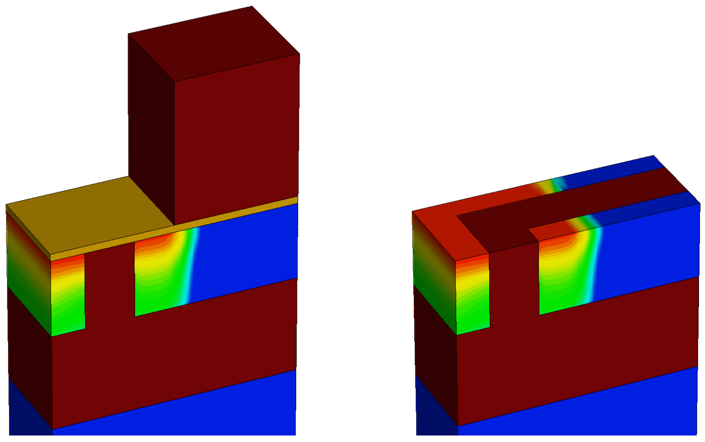


Figure 7.5: Doping via thermal diffusion and hard mask removal

Next, the gate oxide can be deposited. In this example, it was done via a `deposit` command to have a precise thickness, but a dry oxidation can be performed in order to grow a seed layer for other oxide to be deposited on top via CVD. If a high-k dielectric is required, it is possible to do a dry oxidation for a small amount of time in order to have a thin film oxide adhesion layer and then perform atomic layer deposition (ALD) with a specific precursor. For instance, if hafnium dioxide (HfO_2) is needed, the metal-organic amide tetrakis(dimethylamino)hafnium(IV) ($Hf(NMe_2)_4$, TDMAHf) can be used as precursor, often paired with water as the oxygen source. [27] Alternatively, the inorganic precursor hafnium tetrachloride ($HfCl_4$) can also be used, typically with water or ozone as the co-reactant.
Then, a new mask is defined and the oxide is etched to expose the drain contact surface. The figure 7.6 shows that the oxide was also removed everywhere except above the channel. The same mask can be used in step 4 for the deposition of the thick oxide hard mask for channel doping shielding. In this case it is necessary to review the implant and diffusion parameters, as the hard mask will be significantly smaller than before.

```
# -------------------------------------------------------------- #
# STEP 7: GATE OXIDE DEPOSITION                                  #
# -------------------------------------------------------------- #

deposit material= {Oxide} type= anisotropic thickness= @t_gateOx@

mask name= gate_oxides left= -1<nm> right= @L_halfCh@ back= -1<nm>
 ↪  front= @W_halfCh@

etch material= {Oxide} type= anisotropic time= 1<min> rate= {0.00201}
 ↪  mask= gate_oxides
```

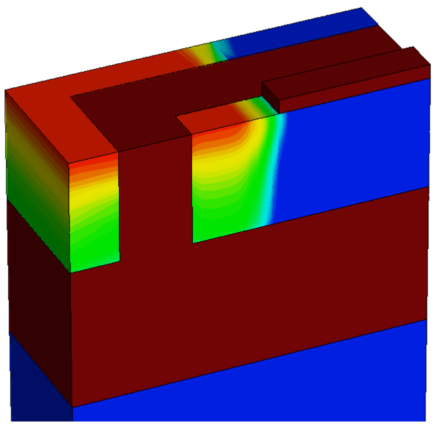


Figure 7.6: Gate oxide deposition

Next, the metal contacts are patterned. For simplicity, a titanium deposition and the definition of etching masks are implemented in SProcess. However, to more accurately model metal gate formation, a replace metal gate (RMG) or damascene process should be used, where a trench is first defined where the contact is to be placed, followed by metal deposition through sputtering and finalized with a CMP step.

```
# ------------------------------------------------------------- #
# STEP 8: CONTACTS DEFINITION                                   #
# ------------------------------------------------------------- #

deposit material= {Titanium} type= anisotropic time= 1<min> rate=
↪   {0.002}

# Middle half barrier gate
mask name= contacts left= -1 right= 2.4<nm> back= -1<nm> front=
↪   @W_halfCh@
# Plunger gate
mask name= contacts left= 4.4<nm> right= 12.4<nm> back= -1<nm> front=
↪   @W_halfCh@
# Second barrier gate
mask name= contacts left= 14.4<nm> right= 19.2<nm> back= -1<nm>
↪   front= @W_halfCh@
# Drain
mask name= contacts left= @L_halfCh@ right= 33.2<nm> back= -1<nm>
↪   front= @W_halfCh@

etch material= {Titanium} type= anisotropic time= 1<min> rate=
↪   {0.00201} mask= contacts
```

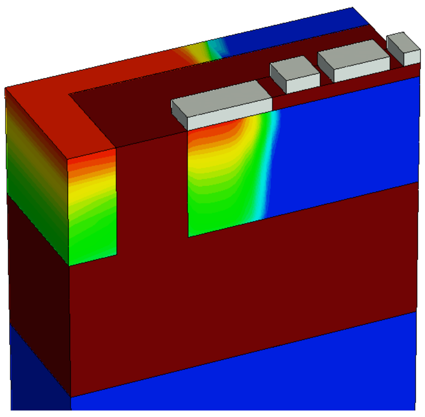


Figure 7.7: Contacts definition

Finally, the geometry can be mirrored twice in order to get the full device. The definition of electrical contacts can be done, but since this device will not be simulated inside SDevice, it is completely optional.

```
# -------------------------------------------------------------- #
# STEP 9: DEVICE MIRRORING                                       #
# -------------------------------------------------------------- #

transform reflect left
transform reflect back

struct tdr= n1_DQD_DEVICE;
```

The final device is shown in figure 7.8. By etching the surface oxide outside the STI, it is possible to observe how the dopant diffuses into the surrounding silicon. This provides an indication of the minimum spacing required for placing the next device on the same wafer.

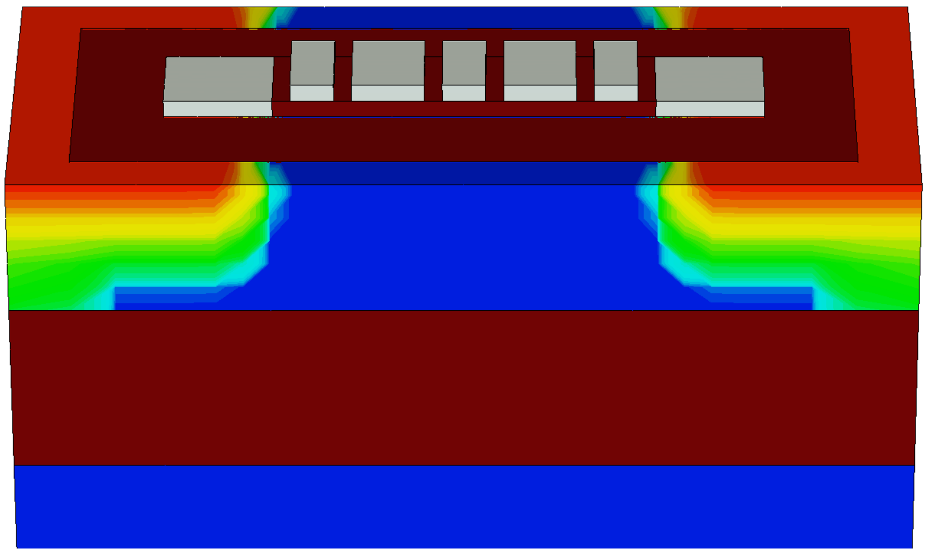


Figure 7.8: Device mirroring and final geometry

7.4 Integration of Doping Profiles into Simulations

It is of particular interest to simulate a device that most closely resembles the one that can be manufactured, especially at PiQuET. One of the most noticeable difference between the two devices is that so far the source and drain contacts were expressed through a perfect rectangular volume. In practice, as outlined in the previous section, the doping concentration is better described as a dopant gradient that penetrates into the silicon. Simulating the device, characterizing this feature, allows for a better understanding of its behaviour in a more realistic scenario. To achieve this, the mesh files and simulation script are modified to incorporate the change. The geometry file must include the doping profiles as separate regions, forming the source and drain contacts, each to be assigned with a distinct value of doping concentration.

7.4.1 Definition Inside Autodesk Fusion

To include the doping profiles in the mesh, the gradient must be defined in Fusion. By importing the `NetActive` profile from Sentaurus as an image and aligning it with the contacts, the overall gradient is approximated using three distinct regions. Additional subdivisions can be introduced to improve the accuracy of the simulation, but increasing the number of volumes also raises file size and computation time. Furthermore, an excessive number of regions seems to cause errors or make the convergence of the Poisson solver more difficult. For this reason, three regions were adopted as a compromise between simulation accuracy and geometry complexity, while ensuring stable convergence. In figure 7.9 it is shown the Fusion sketch, replicating the gradient transitions. The sketch was specifically drawn to distinguish regions that differ by an order of magnitude.

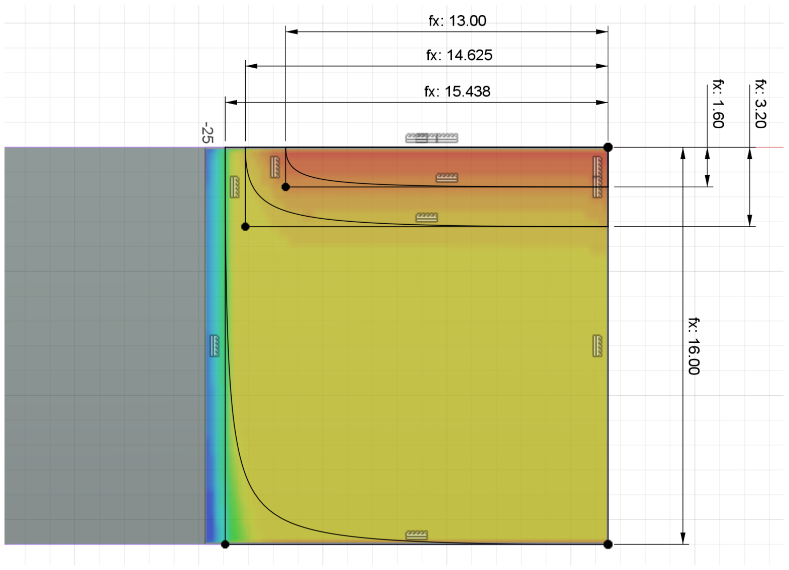


Figure 7.9: Doping profile mesh sketching

Then the sketch can be extruded, forming the new regions of one of the contacts. The outer oxide shell can be reconstructed to resemble the final device. In figure 7.10 these steps are shown. After that, a geometry mirroring with respect to the XoZ plane can be done in order to get the full device.

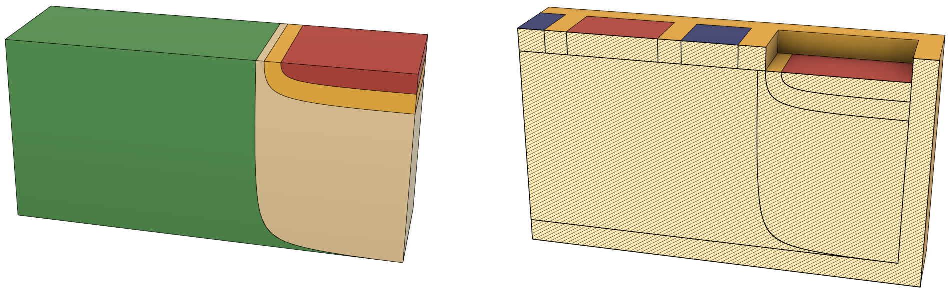


Figure 7.10: Doping profile modelling

The sketching of the doping regions introduced a problem. The geometry now includes a volume that narrows and approaches zero beneath the last gradient region. This causes difficulties when attempting to create a conformal geometry in Gmsh, as it cannot determine where to merge the curves, leading to a failure in processing the geometry. To fix this, the sketch was modified in order to add an indentation in the faulty gradient region, that now takes the place of the narrow region of channel. It is important to note the boolean tolerance set in Gmsh and make the dent bigger that that value. For instance, if the tolerance is set to $1e-3$, the dent should be bigger than $0.001mm$, where millimeters represents the standard measurement unit of the Fusion workspace. The fix is shown in figure 7.11.

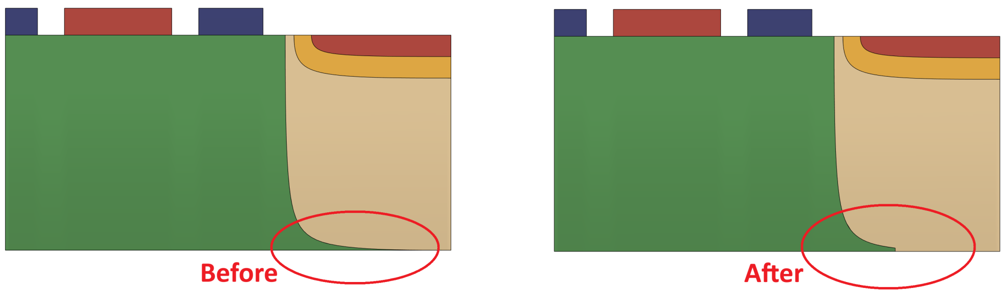


Figure 7.11: Doping profile modelling correction

The geometry can now be exported and used in the Gmsh `.geo` file.

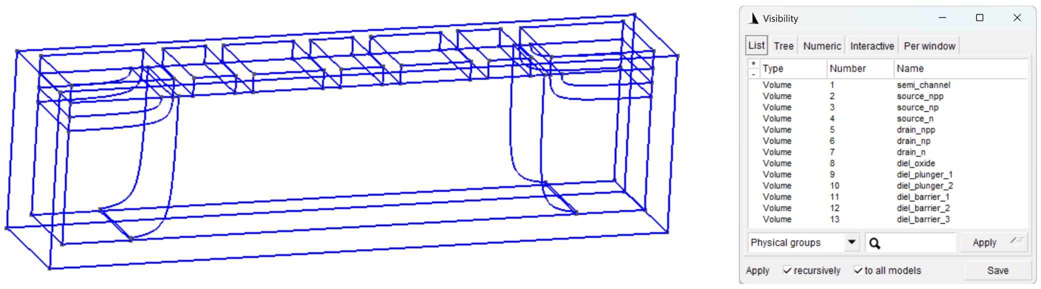
7.4.2 Handling in Gmsh

As for the source and drain definition in the condition of a solid single volume, now the three areas can be labelled to be different regions to be assigned inside the simulation.

```
Physical Volume("source_npp") = {10};
Physical Volume("source_np") = {11};
Physical Volume("source_n") = {9};

Physical Volume("drain_npp") = {2};
Physical Volume("drain_np") = {3};
Physical Volume("drain_n") = {1};
```

The tags `_npp` (n++), `_np` (n+) and `_n` (n) represents the three regions of different doping concentration, starting from the higher one, assigned to the gradient closest to the surface, to the lowest one, forming the interface with the channel.



Type	Number	Name
Volume	1	semi_channel
Volume	2	source_npp
Volume	3	source_np
Volume	4	source_n
Volume	5	drain_npp
Volume	6	drain_np
Volume	7	drain_n
Volume	8	diel_oxide
Volume	9	diel_plunger_1
Volume	10	diel_plunger_2
Volume	11	diel_barrier_1
Volume	12	diel_barrier_2
Volume	13	diel_barrier_3

Figure 7.12: Doping profile handling

The final mesh, with the external oxide hidden in order to show the interior, is shown in figure 7.13. The new physical volumes can now be labelled inside QTCAD to introduce the doping profiles simulation.

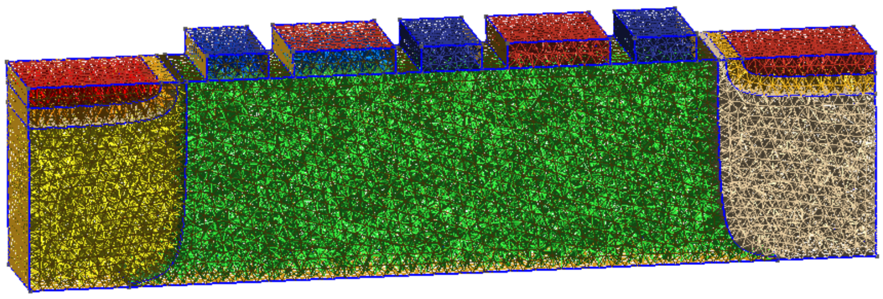


Figure 7.13: Doping profiles meshed, oxide volume is not shown

7.4.3 QTCAD Code Adaptation

The simulation code needs just few modification in order to incorporate the doping profiles. The three doping concentrations needs to be defined. The default value is assigned to the top region, while the others are scaled accordingly to the `NetActive` scale retrieved from Sentaurus. In this case, the regions differs for one order of magnitude, so the other two values are scaled as one tenth and one hundredth of the top concentration.

```
# DEFINE THE DIFFERENT DOPING CONCENTRATIONS TO BE USED. REFER TO
↪  SIMULATION RESULTS FROM SENTAURUS
doping_npp = n_doping      # Define the n_doping as the peak
↪  concentration, localized at the surface
doping_np  = n_doping * 1e-1    # Define the second doping
↪  concentration as one order of magnitude less wrt peak
↪  concentration
doping_n   = n_doping * 1e-2    # Define the third doping
↪  concentration as two order of magnitude less wrt peak
↪  concentration
```

Then, the new regions needs to be labelled with a physical group. Following the tags used in Gmsh, each volume is assigned to the designated doping concentration.

```
d.new_region("source_npp", semiconductor, pdoping=0,
↪  ndoping=doping_npp)
d.new_region("source_np",  semiconductor, pdoping=0,
↪  ndoping=doping_np)
d.new_region("source_n",   semiconductor, pdoping=0,
↪  ndoping=doping_n)

d.new_region("drain_npp",  semiconductor, pdoping=0,
↪  ndoping=doping_npp)
d.new_region("drain_np",   semiconductor, pdoping=0,
↪  ndoping=doping_np)
d.new_region("drain_n",    semiconductor, pdoping=0,
↪  ndoping=doping_n)
```

These two modifications are sufficient to accommodate the new mesh and to simulate the behaviour of the doping profiles through thermal diffusion.

7.5 Silicides

Silicides are compounds formed by the reaction of silicon with metals, commonly used to reduce contact resistance between highly doped silicon and metal contacts. They provide low-resistivity contacts for source, drain and gate regions in CMOS devices, improving current flow and overall device performance. Silicides also enhance thermal stability and reliability of metal-silicon interfaces during high-temperature processing. For this work, the use of a silicide material as a replacement to doped silicon was investigated in order to account for the unavailability of an ion implanter at PiQuET.
To replace the silicon drain, a series of silicidation steps are done in order to make all the silicon underneath react with the metal. These steps include the deposition of titanium, the definition of a mask for etching around the drain, an annealing to make the two materials react to form $TiSi_2$, and finally a CMP to remove excess material.

```
deposit material= {Titanium} type= anisotropic time= 1<min> rate=
 ↪  {0.010}
mask name= silicide left= 16.20<nm> right= 41.2<nm>
etch material= {Titanium} type= anisotropic time= 1<min> rate=
 ↪  {0.0101} mask= silicide

diffuse temperature= 500<C> time= 120<s>
etch type= cmp coord= 0.0  material= all
```

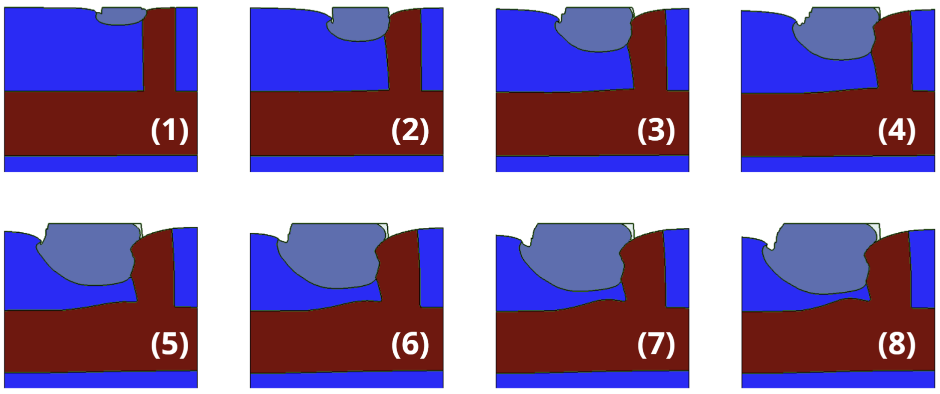


Figure 7.14: Silicidation steps

As shown in figure 7.14, the silicidation steps introduces a structure deformation that cannot be controlled. This negatively affects the reproducibility of the simulation results. Furthermore, the modelling of the material, as it is or by using an equivalent doped silicon, need to be addressed in a precise way in order to include the effects of this fabrication step inside the quantum simulations. These two issues present a significant barrier, making the idea of using silicides as a substitute for doped silicon contacts impractical. Consequently, this approach was discarded.

7.6 Barrier Gate Manufacturing

The need for such a small device led to challenges in defining the gate metallizations. In particular, the smallest gates, the barrier gates, may be below the resolution limit of the EBL, making them difficult to fabricate. To address this issue, a different approach was investigated. The idea is to define source, drain and plunger gate contacts as usual and then exploit the oxidation of metal to create a dielectric barrier to accommodate and isolate the barrier gates contacts, realized by depositing another metal layer and then performing a CMP. In order to oxidate metals, an electrochemical surface treatment called Plasma Enhanced Oxidation is employed. It is similar to anodizing, but it operates at much higher voltages, leading to micro-discharges that generate a plasma capable of modifying the oxide layer's structure. This technique enables the formation of thick (ranging from tens to hundreds of micrometers) and predominantly crystalline oxide coatings, particularly on metals such as aluminium, magnesium and titanium. [10] This process could not be simulated inside Sentaurus Process with ease, so it was modelled with an oxide deposition. The main drawback is that with a metal oxidation, the oxide starts forming from the metal contact, growing perpendicularly to the surface, while by depositing an oxide layer it adds to the thickness seen by the barrier-silicon gate capacitance. This could not be an issue if correctly modelled inside the quantum simulations. The main fabrication steps, are shown in figure 7.15.

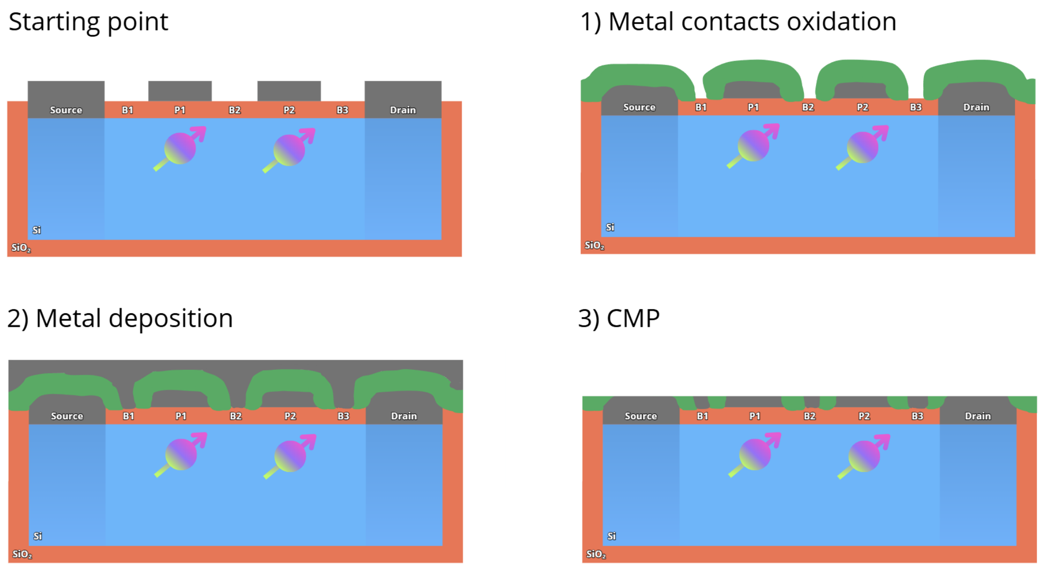


Figure 7.15: Barrier gates manufacturing steps

The main SProcess code is now analyzed, accounting for an oxide deposition as a replacement for plasma enhanced oxidation.

The code follows the one explained in Section 7.3, just before the contacts definition. In this case, the definition of the barrier gates is not performed. As explained in Section 7.3, metal gate fabrication is modelled as a straightforward deposition and etching step for simplicity. A more realistic approach would involve damascene processing.

```
# --------------------------------------------------------------- #
# STEP 1: PLUNGER AND DRAIN CONTACTS                              #
# --------------------------------------------------------------- #

deposit material= {Titanium} type= anisotropic thickness= 5.0<nm>

mask name= plungerANDdrain1 left= 4.4<nm> right= 12.4<nm>  back=
↪  -1<nm> front= 5.0<nm>
mask name= plungerANDdrain1 left= 21.2<nm> right= 33.2<nm>  back=
↪  -1<nm> front= 5.0<nm>

etch material= {Titanium} type= anisotropic thickness= 5.01<nm> mask=
↪  plungerANDdrain1
```

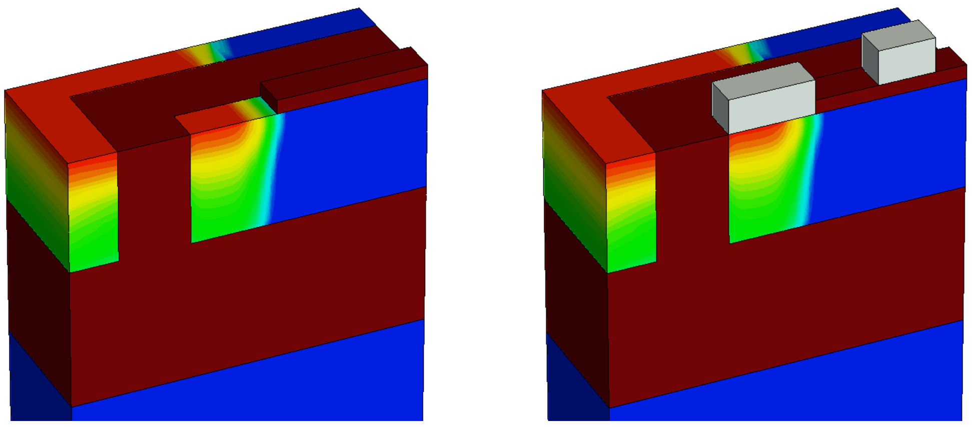


Figure 7.16: Gate definition

Next, metal oxidation is carried out to form a dielectric layer that insulates the contacts while simultaneously creating a small trench for barrier gate deposition. Since metal oxidation is not easily achievable in SProcess, the process was emulated by depositing a placeholder dielectric, in this case silicon dioxide, as a substitute for the metal oxide. To support this approach, a mask slightly smaller than the gate contacts is used to ensure lateral deposition. The isotropic deposition effectively replicates the behaviour of metal oxide growth, protruding perpendicularly from the metal gate surface. If an anisotropic deposition is used instead, it is necessary to keep in mind that the oxide layer will grow onto the existing trench, adding to the total barrier gate oxide thickness. Another approach could involve etching the oxide using the active area mask, with the oxide beneath the drain and plunger gate protected by the metal, followed by oxide deposition. However, the absence of a seed oxide layer could pose challenges for this method. Therefore, an isotropic deposition was performed.

```
# -------------------------------------------------------------- #
# STEP 2: OXIDATION                                              #
# -------------------------------------------------------------- #

mask name= plungerANDdrain2 left= 1.27<nm> right= 15.4<nm>  back=
↪  -1<nm> front= 8.0<nm> negative
mask name= plungerANDdrain2 left= 18.2<nm> right= 36.2<nm>  back=
↪  -1<nm> front= 8.0<nm> negative

deposit material= {Oxide} type= isotropic time= 1<min> rate= {0.004}
↪  mask= plungerANDdrain2
```

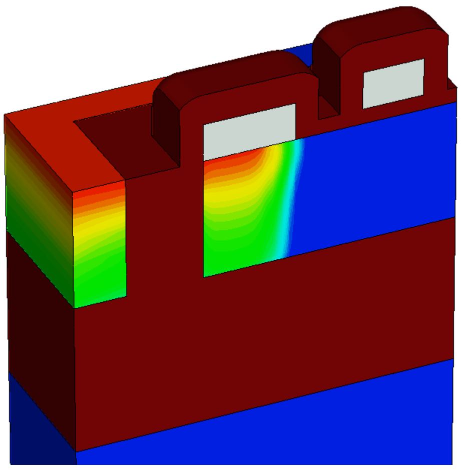


Figure 7.17: Plasma Enhanced Oxidation

Next, the titanium is deposited via sputtering. The direction vector along the multiple etching steps are performed in order to emulate the directional deposition of those physical vapour deposition (PVD) machines, which is typically around $20 \div 30°$ of tilt.

```
# ------------------------------------------------------------ #
# STEP 3: TITANIUM DEPOSITION                                   #
# ------------------------------------------------------------ #

refinebox Titanium min= {-14.0 0.0 0.0} max= {0.0 0.0332 0.006}
 ↪  xrefine= {0.001 0.001 0.001} yrefine= {0.001 0.001 0.001}
 ↪  zrefine= {0.001 0.001 0.001} add
grid remesh

deposit material= {Titanium} type= directional thickness= 8.0<nm>
 ↪  direction= {0.5 1 1}

mask name= barriers left= -1<nm> right= 21.2<nm> back= -1<nm> front=
 ↪  5.0<nm>
etch material= {Titanium} type= directional thickness= 8.01<nm>
 ↪  direction= {0.5 1 1} mask= barriers
etch material= {Titanium} type= directional thickness= 8.01<nm>
 ↪  direction= {0.5 -1 1} mask= barriers
etch material= {Titanium} type= directional thickness= 8.01<nm>
 ↪  direction= {0.5 -1 -1} mask= barriers
etch material= {Titanium} type= directional thickness= 8.01<nm>
 ↪  direction= {0.5 1 -1} mask= barriers
```

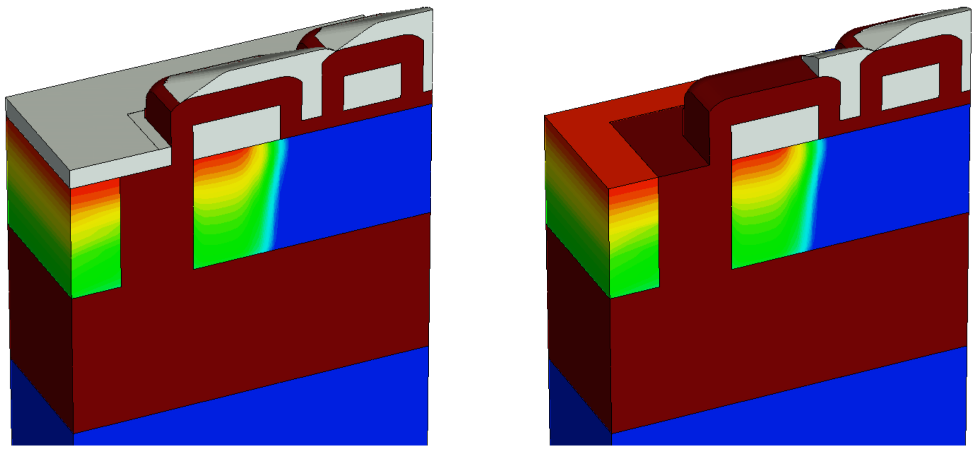


Figure 7.18: Metal deposition and directional etching

Finally, a thick layer of oxide is deposited and a final CMP planarizes the whole device. The geometry is mirrored twice to obtain the final device, as shown in figure 7.19.

```
# ------------------------------------------------------------ #
# STEP 4: PLANARIZATION + CMP                                  #
# ------------------------------------------------------------ #

deposit material= {Oxide} type= anisotropic time= 1<min> rate=
↪  {0.014}

etch type= cmp coord= -0.004  material= all

# ------------------------------------------------------------ #
# STEP 5: GEOMETRY MIRRORING                                   #
# ------------------------------------------------------------ #

transform reflect left
transform reflect back

struct tdr= n1_DQD_COMPLETE;
```

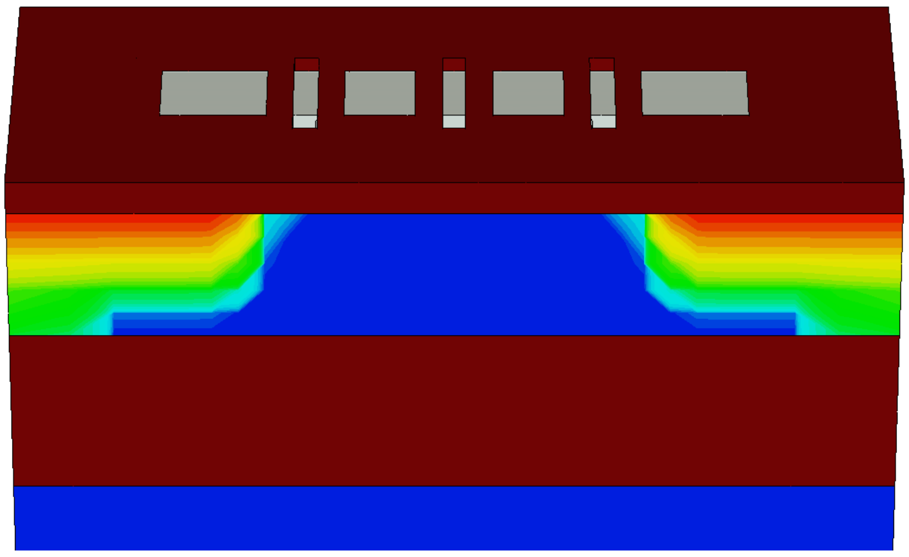


Figure 7.19: CMP + Mirroring

Chapter 8

Experimental Results from QTCAD

In this chapter, the main simulation results from QTCAD for each device are presented and discussed. While the transport simulations were carried out manually by selecting a promising configuration, setting the input parameters and starting the computation, the confinement simulations were performed with a modified version of the batch runner, enabling the simultaneous processing of all devices. This modified runner script reduces human intervention, as each device simulation can take up to two hours and manually changing input configurations between devices is time consuming. The runner was left to operate autonomously overnight, completing the simulation of all devices in approximately nine hours, while notifying via email about the progress, as explained in Chapter 6. It automatically compute, for each device, the lever arm curves for the first four energetic states, the ground state lever arm sweep for different values of applied voltage on the first plunger gate and the computation of confinement defined on

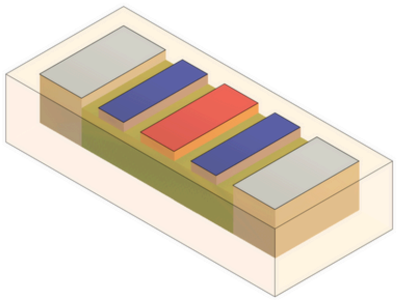
- `plunger_values = [0.5, 0.6, 0.8, 1.0, 1.2]`
- `doping_values = [1e15*1e6,1e16*1e6,1e17*1e6,1e18*1e6,1e19*1e6,1e20*1e6]`
- `barrier_values = [-0.1, -0.2, -0.3]`

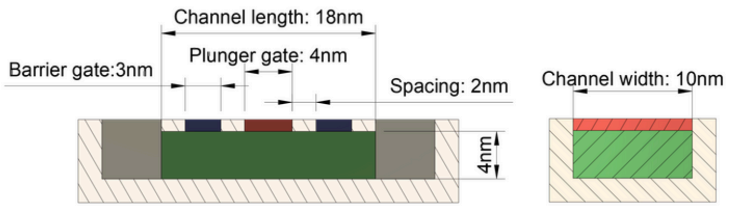
giving a total of $5 \cdot 6 \cdot 3 = 90$ simulation configurations for each device. Given the high number of results, this section shows only six of them for each device: some were selected based on their score and others to highlight defective behaviour in certain configurations, such as wavefunction asymmetry in double quantum dot devices.

For three-dimensional devices, the wavefunctions were cropped around the channel (such as the top section of a fin or the center of a nanowire) rather than the entire device, excluding regions that do not contribute to the results and would otherwise produce cluttered or less informative plots. For this reason, the z-axis cumulative confinement levels for nanowire devices needs to be interpreted as the bottom half of the nanowire. For instance, if 6σ is at $-2nm$, the dot is 99.9999% confined in a $4nm$ region centred at the dot origin.

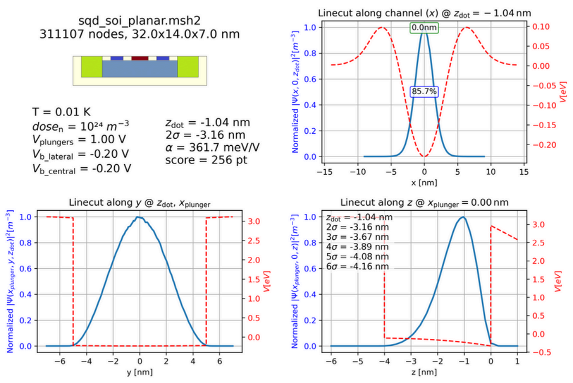
Structure 1
sqd_soi_planar

Single Quantum Dot
SOI Planar

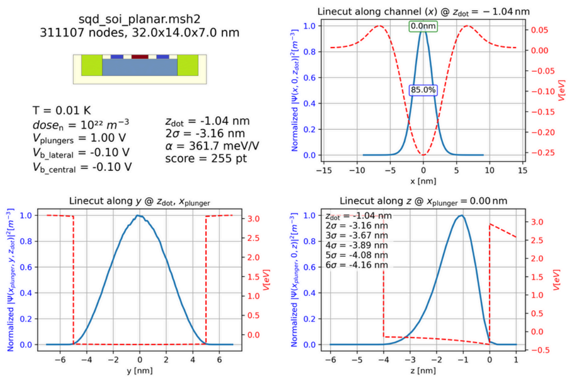




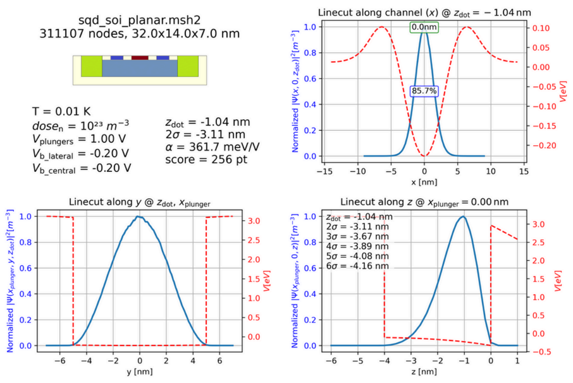
Confinement results



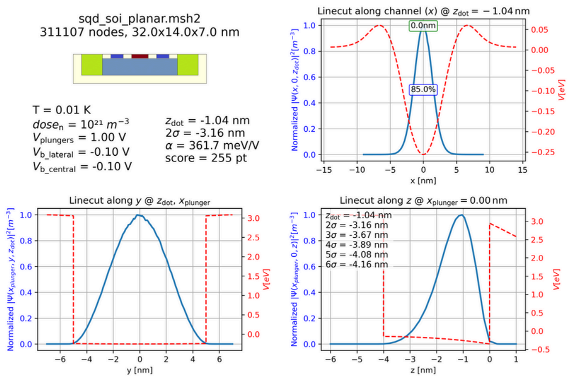
T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.20 V
$V_{b\_central}$ = -0.20 V
z_{dot} = -1.04 nm
2σ = -3.16 nm
α = 361.7 meV/V
score = 256 pt



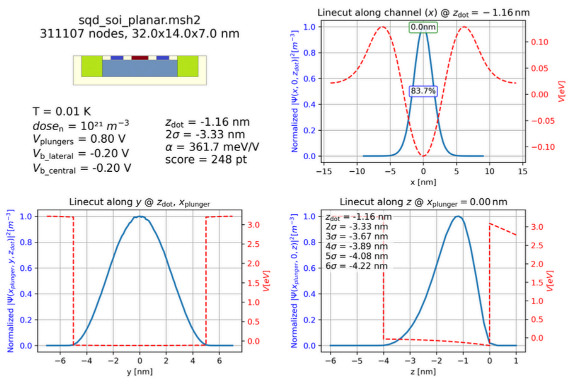
T = 0.01 K
$dose_n = 10^{22}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V
z_{dot} = -1.04 nm
2σ = -3.16 nm
α = 361.7 meV/V
score = 255 pt



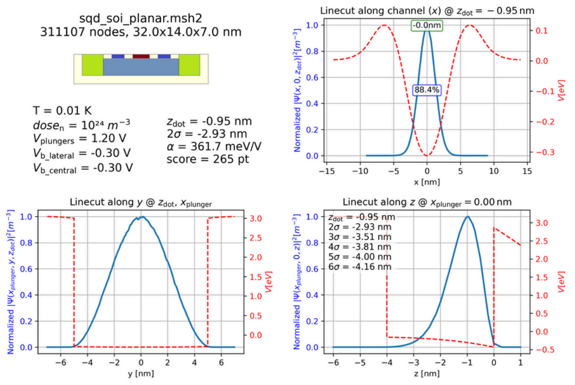
T = 0.01 K
$dose_n = 10^{23}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.20 V
$V_{b\_central}$ = -0.20 V
z_{dot} = -1.04 nm
2σ = -3.11 nm
α = 361.7 meV/V
score = 256 pt



T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V
z_{dot} = -1.04 nm
2σ = -3.16 nm
α = 361.7 meV/V
score = 255 pt

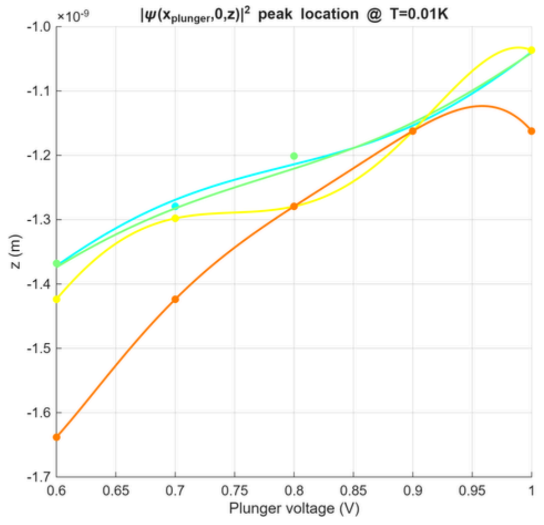


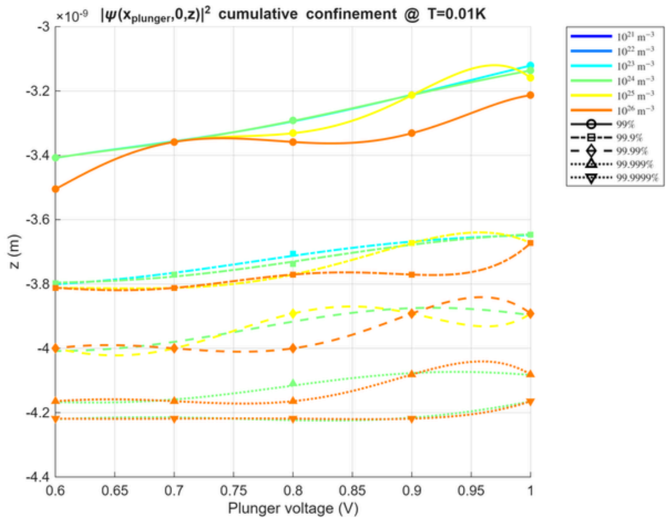
T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 0.80 V
$V_{b\_lateral}$ = -0.20 V
$V_{b\_central}$ = -0.20 V
z_{dot} = -1.16 nm
2σ = -3.33 nm
α = 361.7 meV/V
score = 248 pt

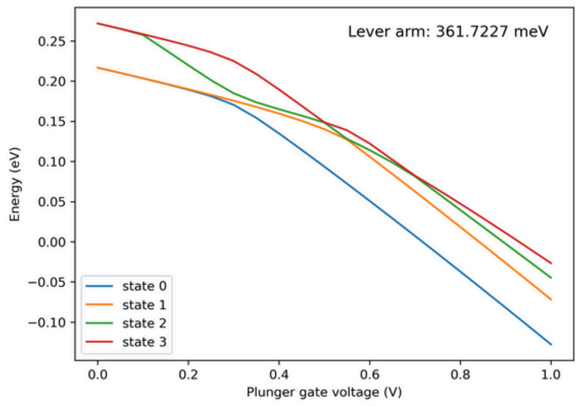


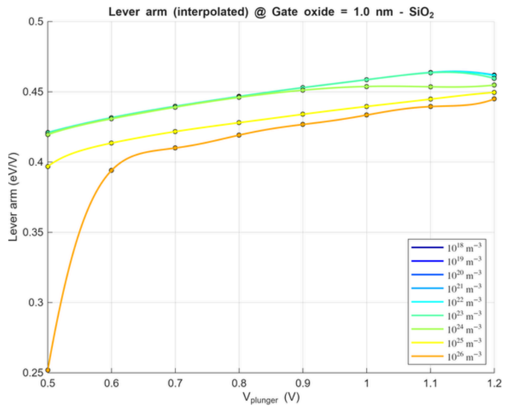
T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V
z_{dot} = -0.95 nm
2σ = -2.93 nm
α = 361.7 meV/V
score = 265 pt

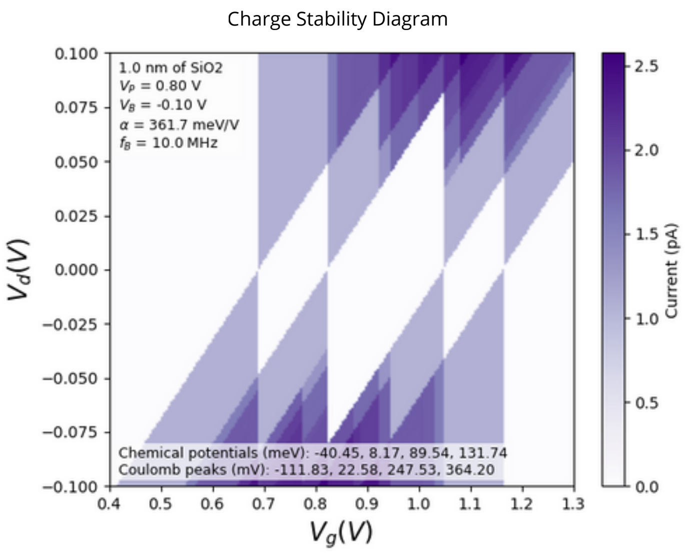
Transport results





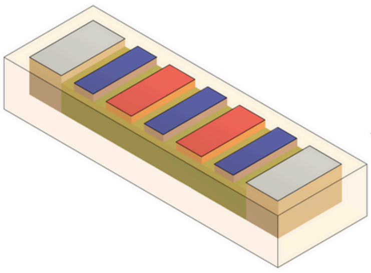


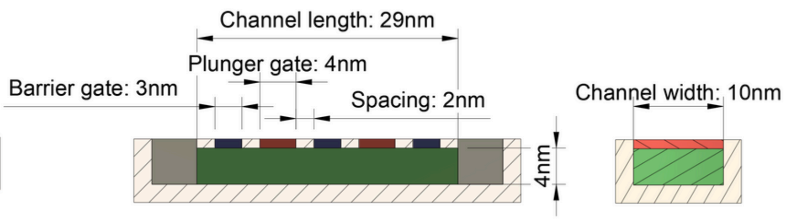




Structure 2
dqd_soi_planar

Double Quantum Dot
SOI Planar



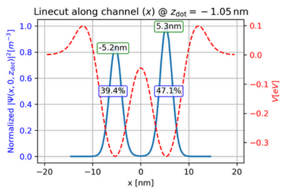


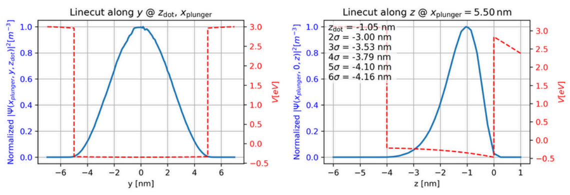
Confinement results

dqd_soi_planar.msh2
416950 nodes, 43.0x14.0x7.0 nm

T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = -1.05 nm
2σ = -3.00 nm
α = 204.9 meV/V
score = 194 pt

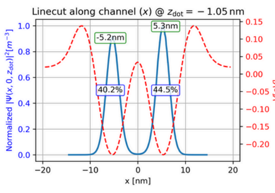


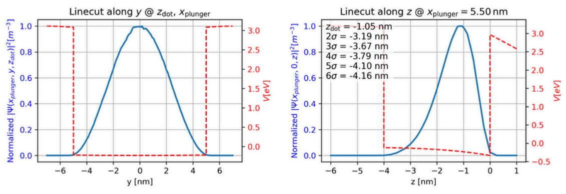


dqd_soi_planar.msh2
416950 nodes, 43.0x14.0x7.0 nm

T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = -1.05 nm
2σ = -3.19 nm
α = 204.9 meV/V
score = 172 pt

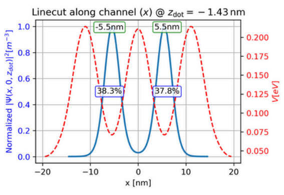


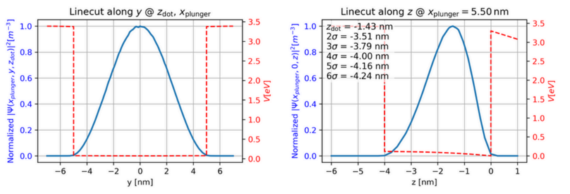


dqd_soi_planar.msh2
416950 nodes, 43.0x14.0x7.0 nm

T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 0.50 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = -1.43 nm
2σ = -3.51 nm
α = 204.9 meV/V
score = 170 pt

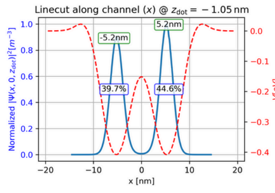


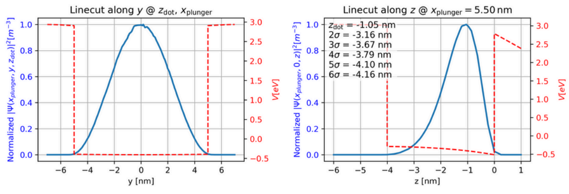


dqd_soi_planar.msh2
416950 nodes, 43.0x14.0x7.0 nm

T = 0.01 K
$dose_n = 10^{22}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

z_{dot} = -1.05 nm
2σ = -3.16 nm
α = 204.9 meV/V
score = 163 pt

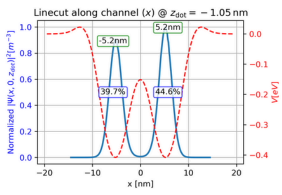


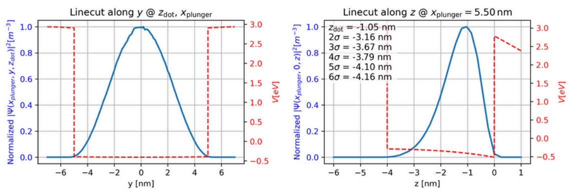


dqd_soi_planar.msh2
416950 nodes, 43.0x14.0x7.0 nm

T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

z_{dot} = -1.05 nm
2σ = -3.16 nm
α = 204.9 meV/V
score = 163 pt

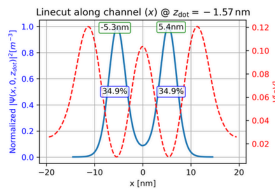


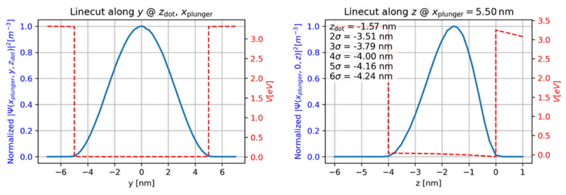


dqd_soi_planar.msh2
416950 nodes, 43.0x14.0x7.0 nm

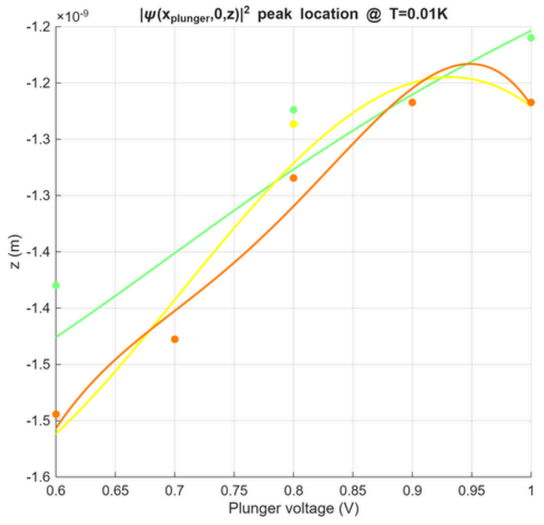
T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 0.50 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

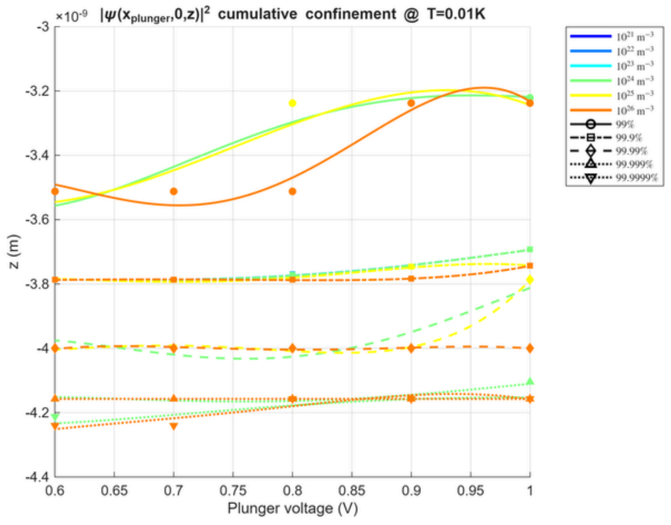
z_{dot} = -1.57 nm
2σ = -3.51 nm
α = 204.9 meV/V
score = 102 pt

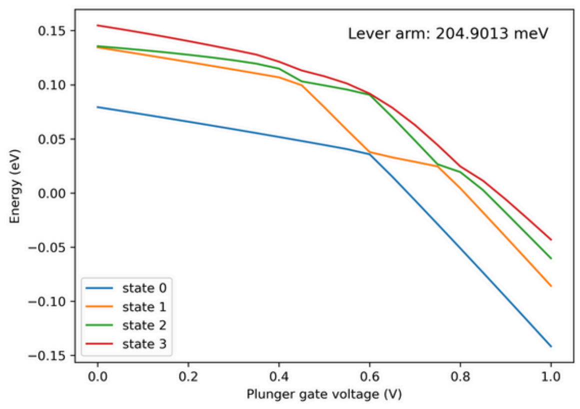


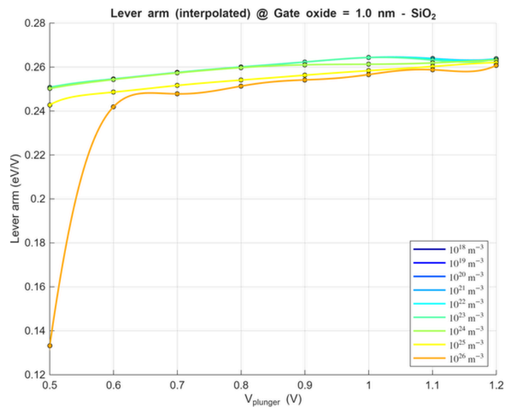


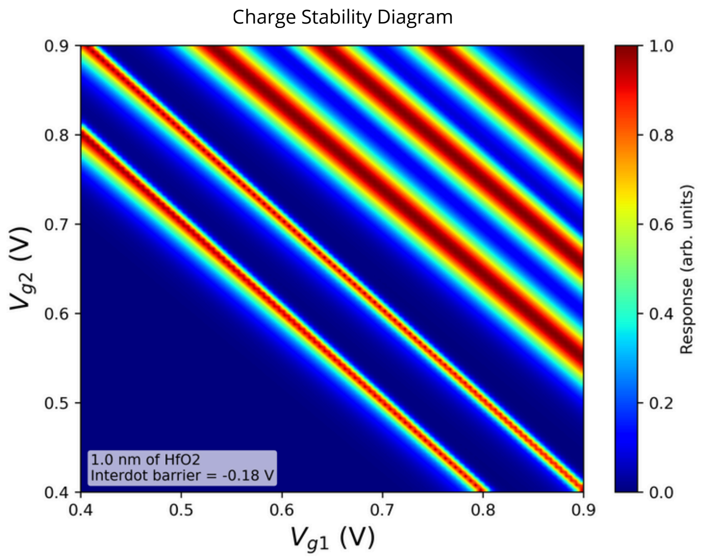
Transport results





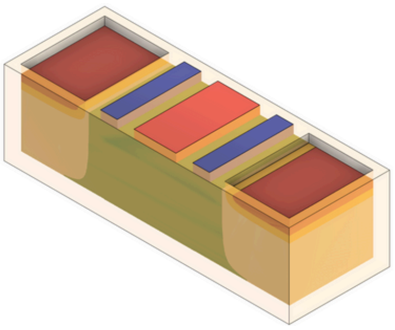


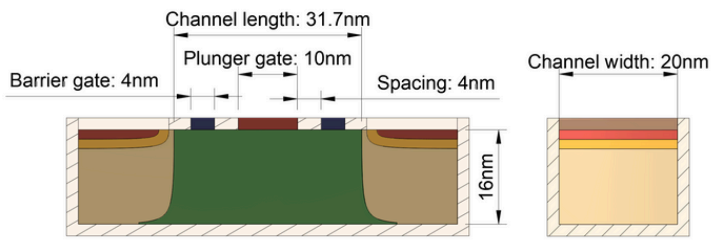




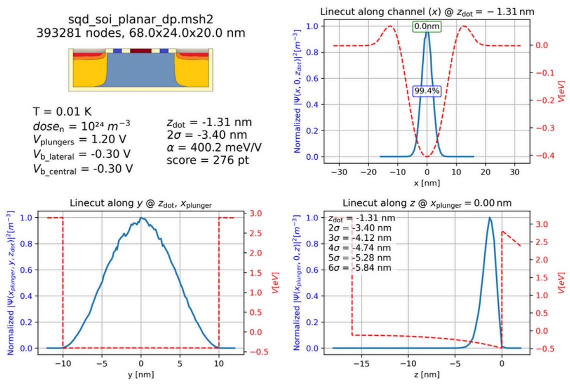
Structure 3
sqd_soi_planar_dp

Double Quantum Dot
SOI Planar
Doping profiles modelled

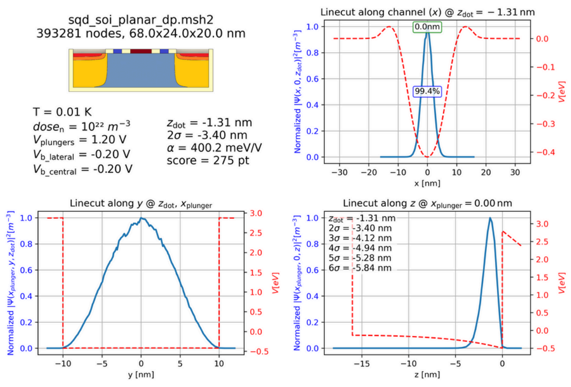




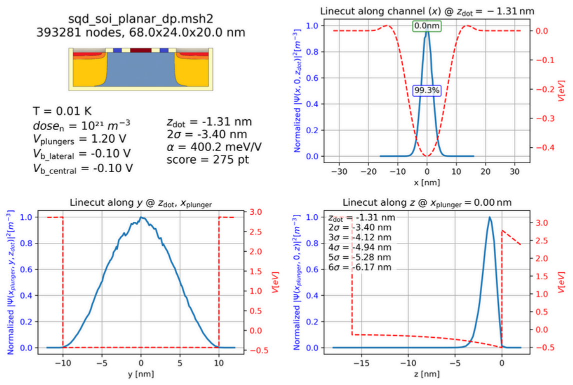
Confinement results



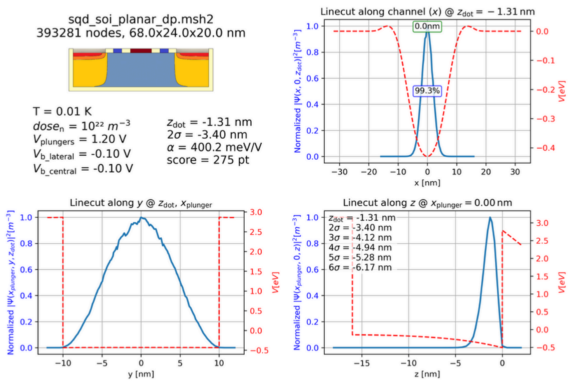
T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V
z_{dot} = -1.31 nm
2σ = -3.40 nm
α = 400.2 meV/V
score = 276 pt



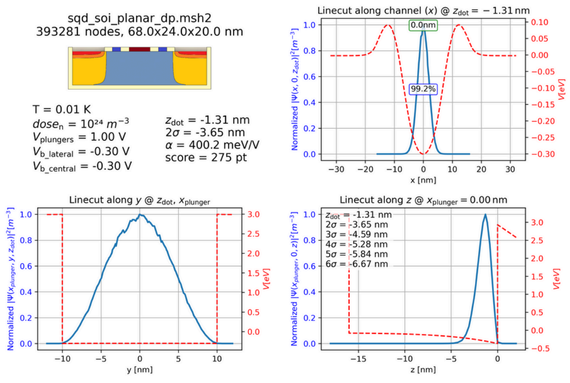
T = 0.01 K
$dose_n = 10^{22}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.20 V
$V_{b\_central}$ = -0.20 V
z_{dot} = -1.31 nm
2σ = -3.40 nm
α = 400.2 meV/V
score = 275 pt



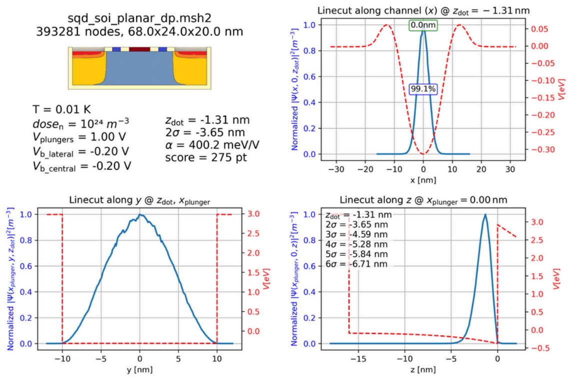
T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V
z_{dot} = -1.31 nm
2σ = -3.40 nm
α = 400.2 meV/V
score = 275 pt



T = 0.01 K
$dose_n = 10^{22}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V
z_{dot} = -1.31 nm
2σ = -3.40 nm
α = 400.2 meV/V
score = 275 pt

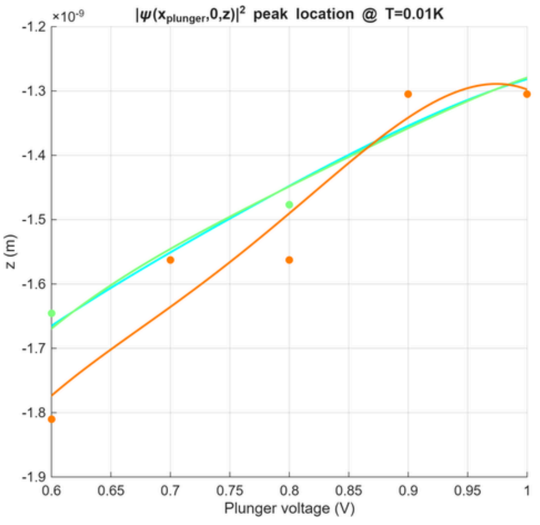


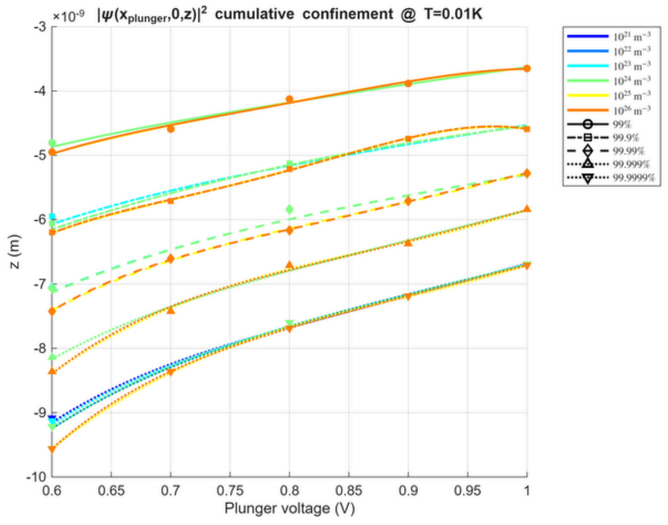
T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V
z_{dot} = -1.31 nm
2σ = -3.65 nm
α = 400.2 meV/V
score = 275 pt

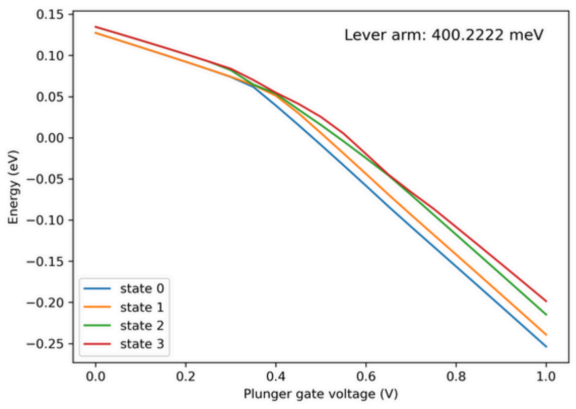


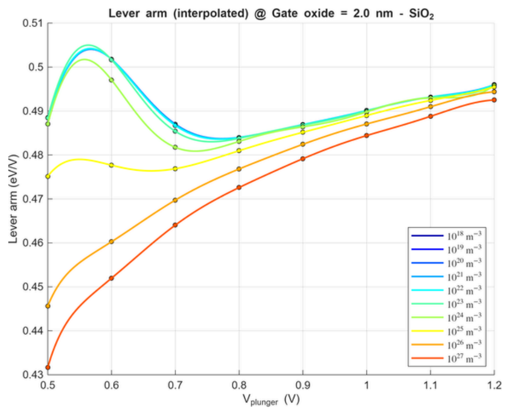
T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.20 V
$V_{b\_central}$ = -0.20 V
z_{dot} = -1.31 nm
2σ = -3.65 nm
α = 400.2 meV/V
score = 275 pt

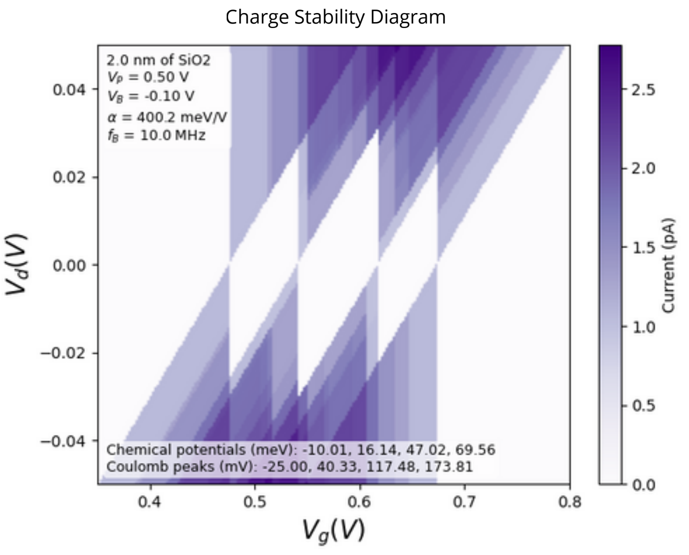
Transport results





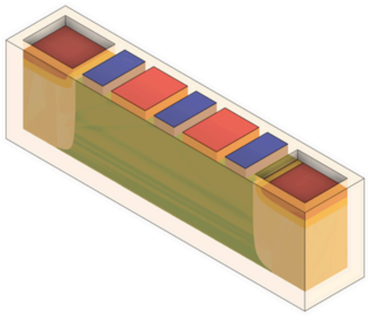


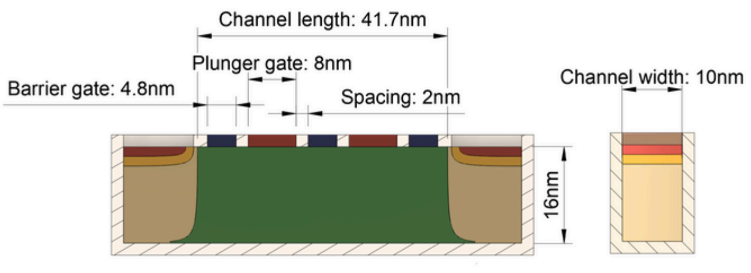




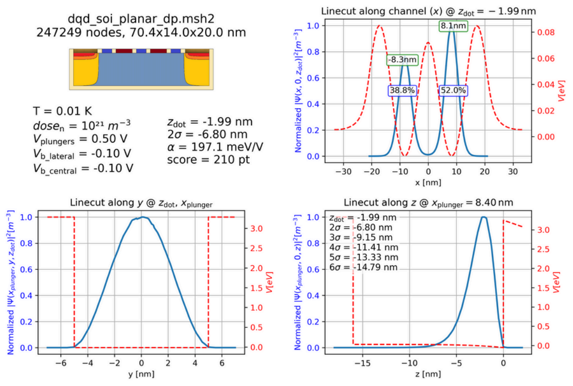
Structure 4
dqd_soi_planar_dp

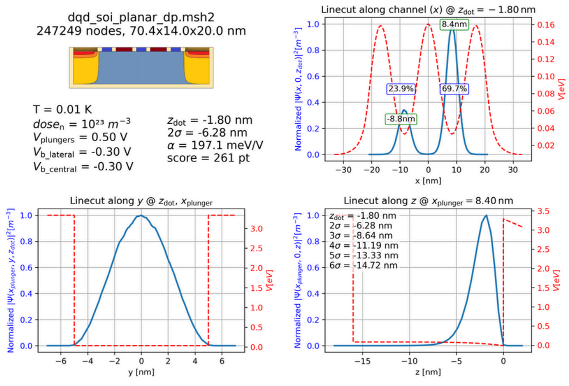
Single Quantum Dot
SOI Planar
Doping profiles modelled

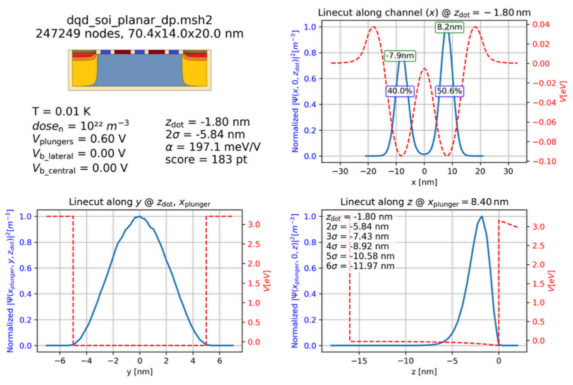


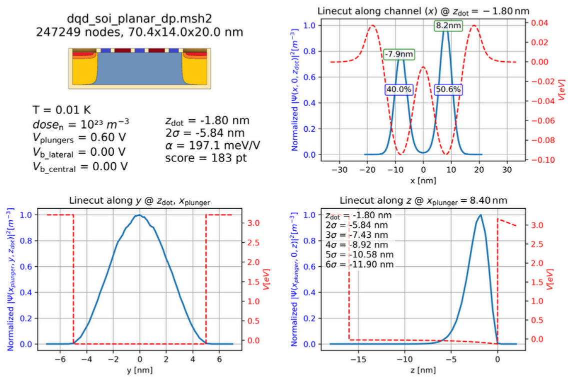


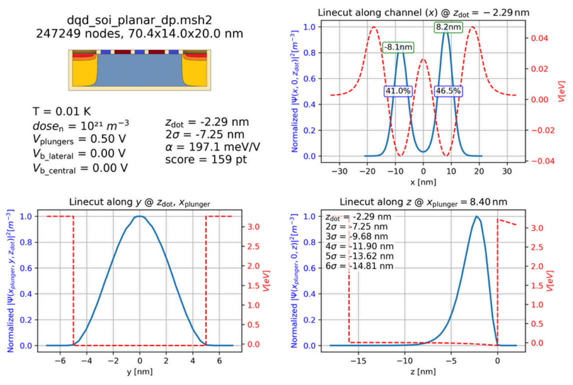
Confinement results

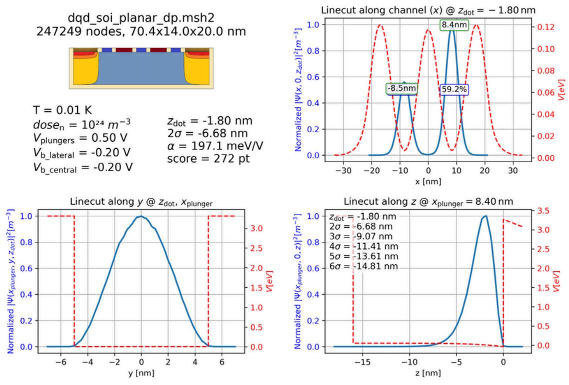




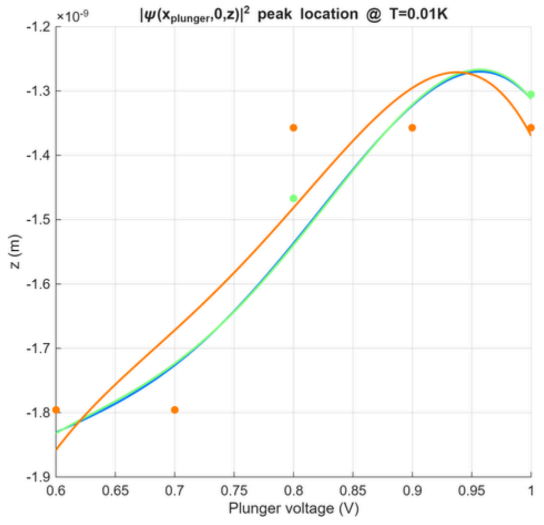


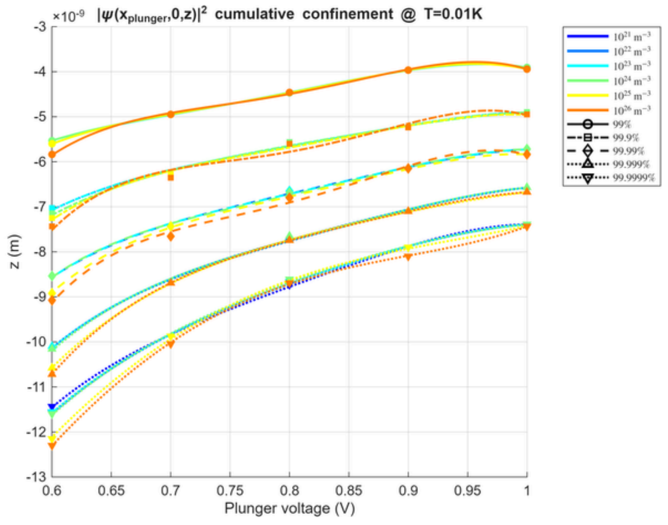


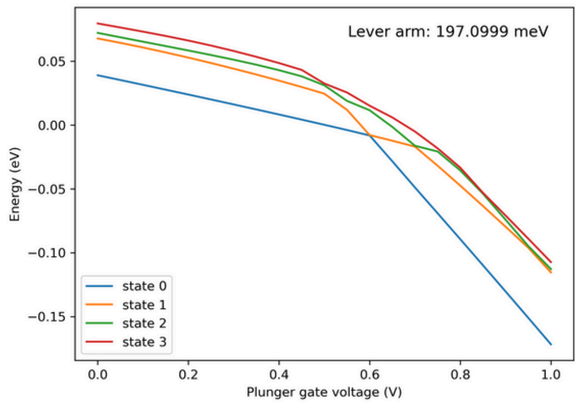


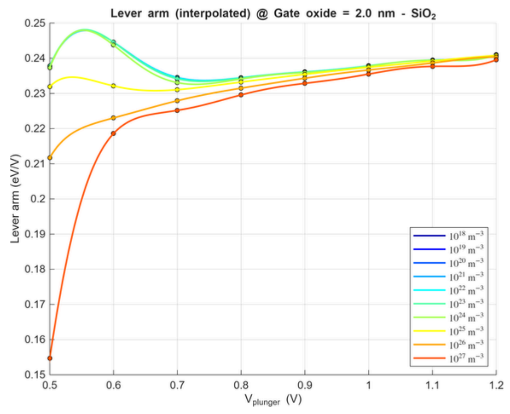


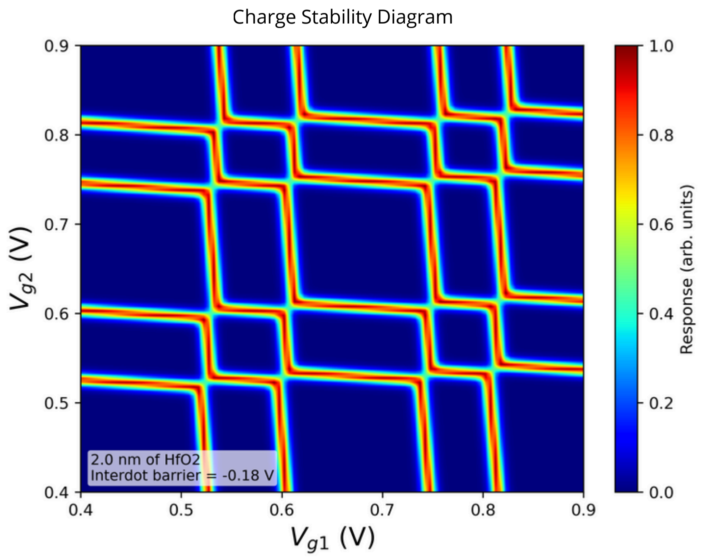
Transport results





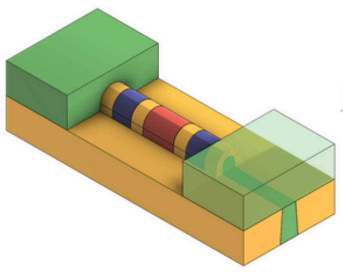


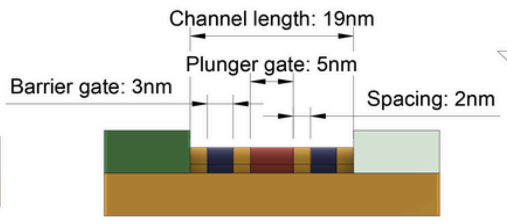


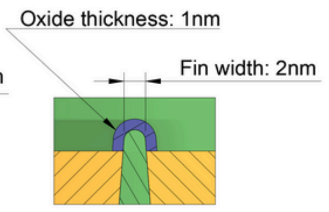


Structure 5
sqd_soi_finfet

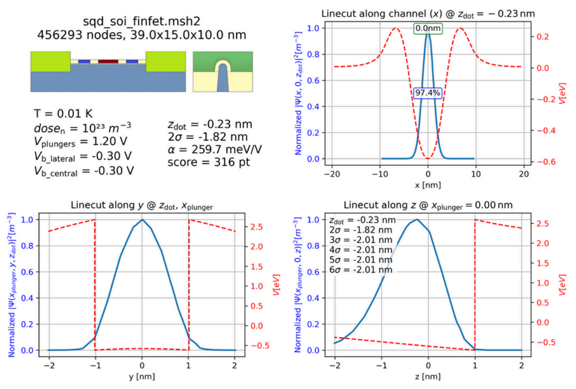
Single Quantum Dot
SOI FinFET-like
Simulated fin tip only





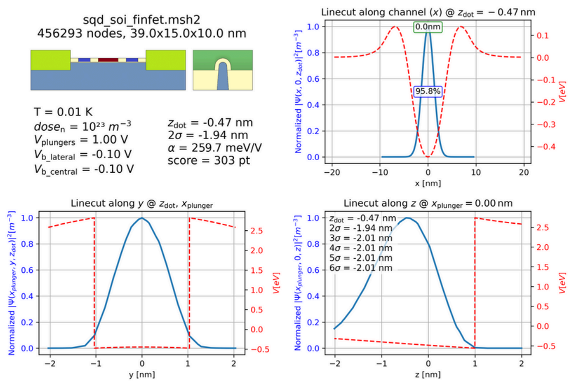


Confinement results



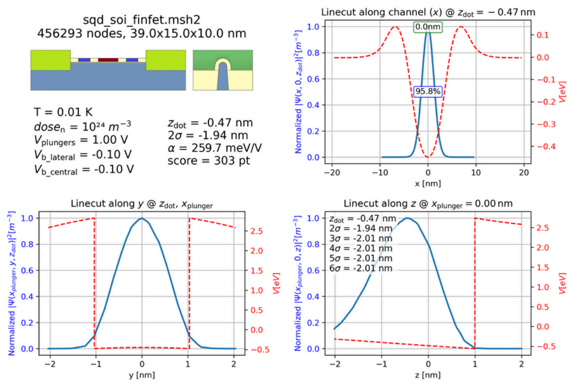
T = 0.01 K
$dose_n = 10^{23}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = -0.23 nm
2σ = -1.82 nm
α = 259.7 meV/V
score = 316 pt



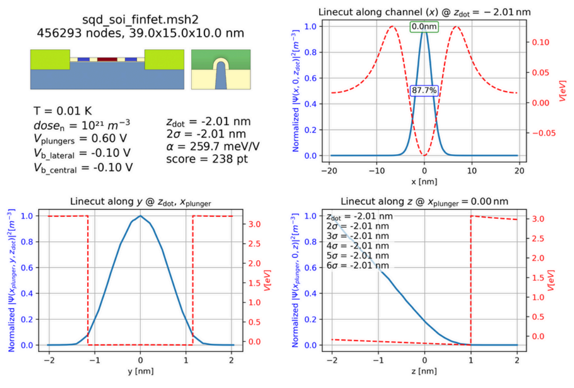
T = 0.01 K
$dose_n = 10^{23}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

z_{dot} = -0.47 nm
2σ = -1.94 nm
α = 259.7 meV/V
score = 303 pt



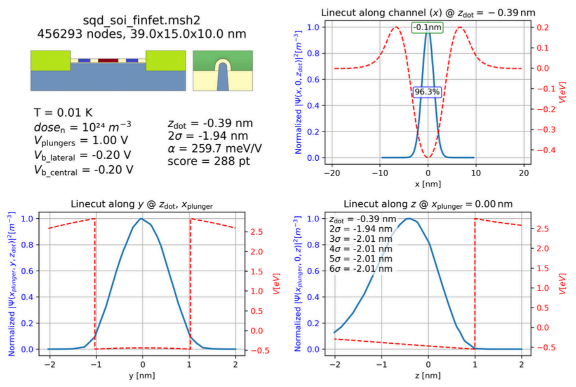
T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

z_{dot} = -0.47 nm
2σ = -1.94 nm
α = 259.7 meV/V
score = 303 pt



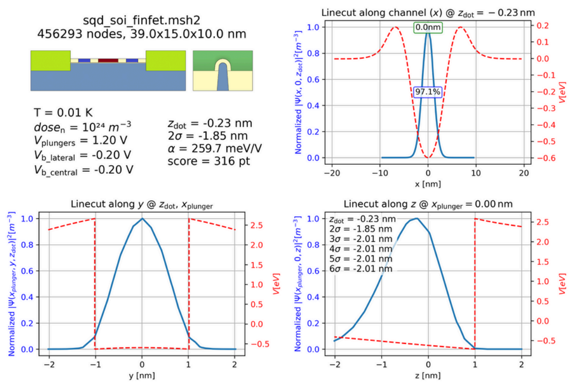
T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 0.60 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

z_{dot} = -2.01 nm
2σ = -2.01 nm
α = 259.7 meV/V
score = 238 pt



T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.20 V
$V_{b\_central}$ = -0.20 V

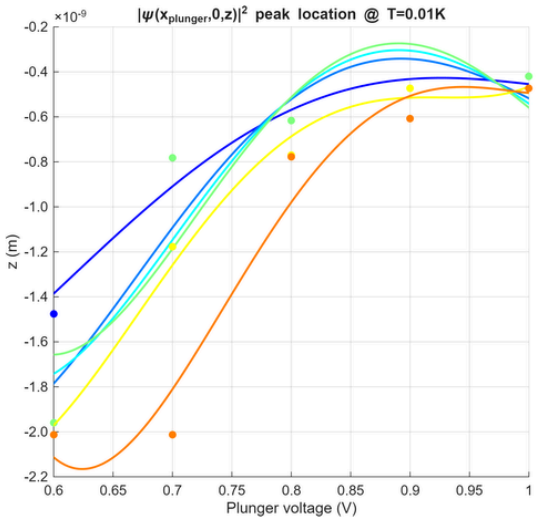
z_{dot} = -0.39 nm
2σ = -1.94 nm
α = 259.7 meV/V
score = 288 pt

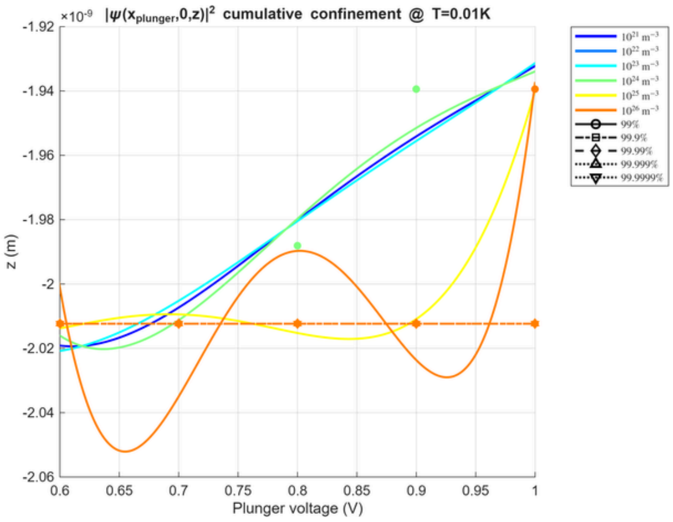


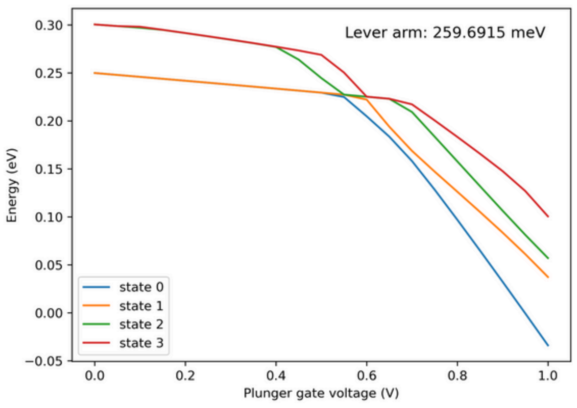
T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.20 V
$V_{b\_central}$ = -0.20 V

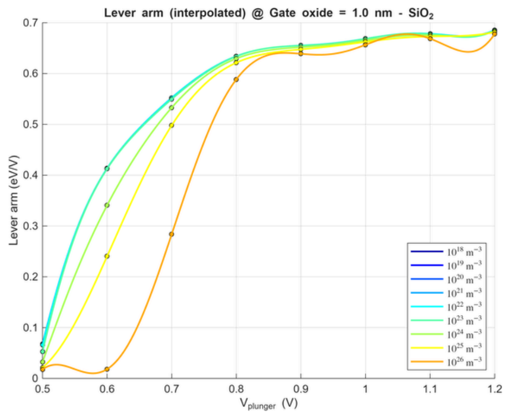
z_{dot} = -0.23 nm
2σ = -1.85 nm
α = 259.7 meV/V
score = 316 pt

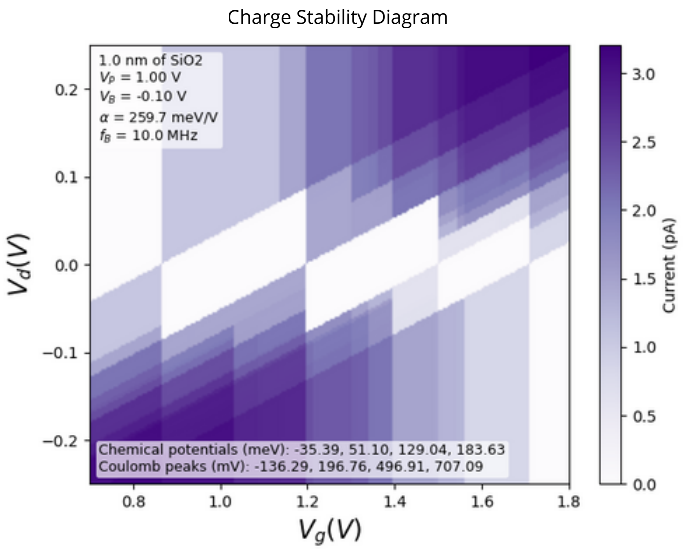
Transport results





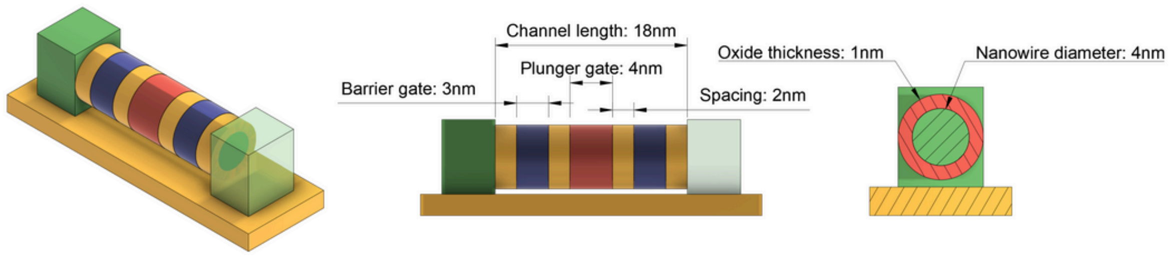




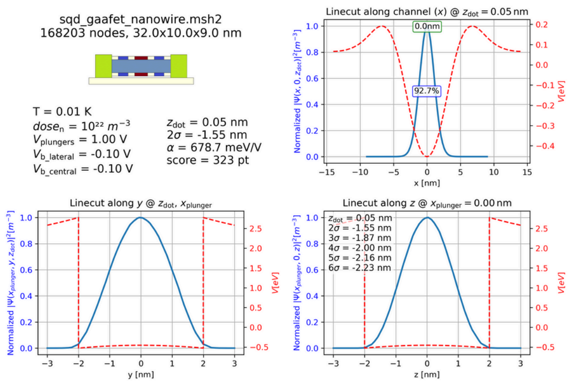


Structure 6
sqd_gaafet_nanowire

Single Quantum Dot
Gate All Around Nanowire
Substrate and contacts defined

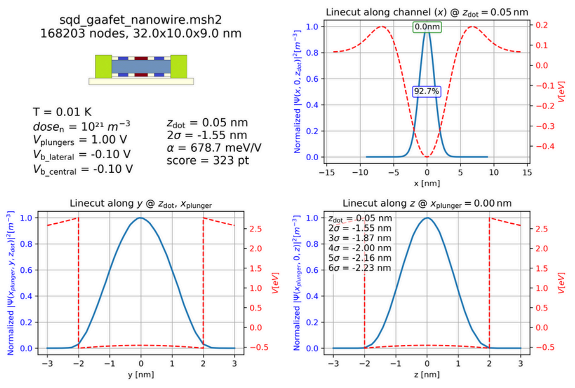


Confinement results


sqd_gaafet_nanowire.msh2
168203 nodes, 32.0x10.0x9.0 nm

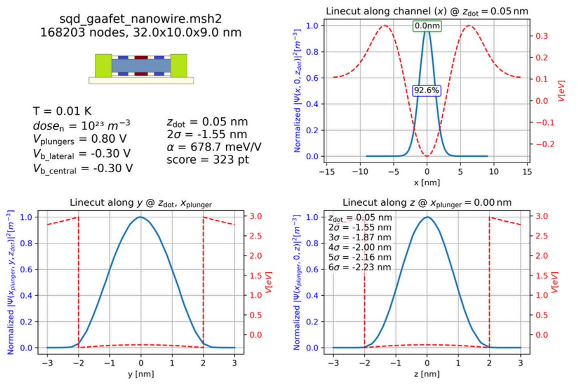
T = 0.01 K
$dose_n = 10^{22}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

z_{dot} = 0.05 nm
2σ = -1.55 nm
α = 678.7 meV/V
score = 323 pt


sqd_gaafet_nanowire.msh2
168203 nodes, 32.0x10.0x9.0 nm

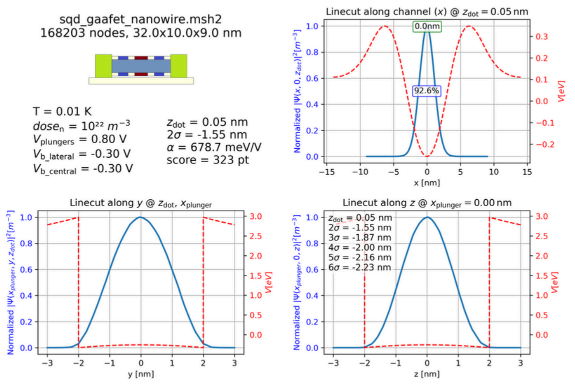
T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 1.00 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

z_{dot} = 0.05 nm
2σ = -1.55 nm
α = 678.7 meV/V
score = 323 pt


sqd_gaafet_nanowire.msh2
168203 nodes, 32.0x10.0x9.0 nm

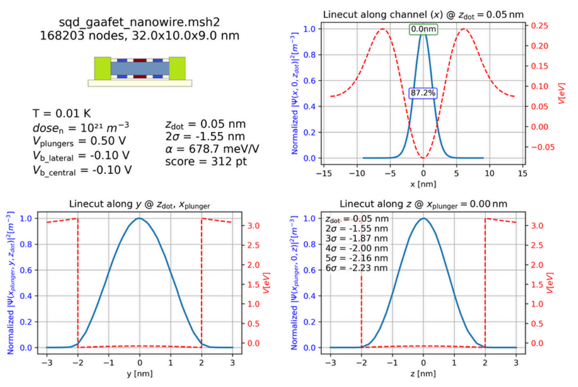
T = 0.01 K
$dose_n = 10^{23}\ m^{-3}$
$V_{plungers}$ = 0.80 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = 0.05 nm
2σ = -1.55 nm
α = 678.7 meV/V
score = 323 pt


sqd_gaafet_nanowire.msh2
168203 nodes, 32.0x10.0x9.0 nm

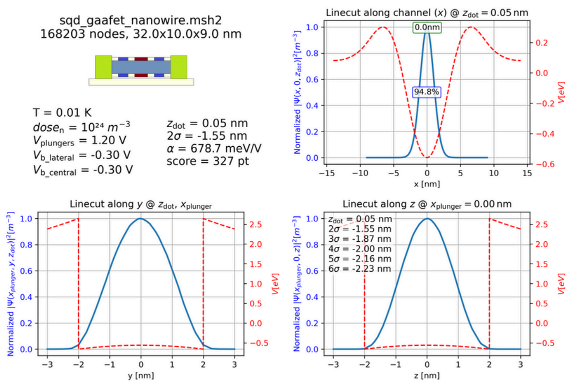
T = 0.01 K
$dose_n = 10^{22}\ m^{-3}$
$V_{plungers}$ = 0.80 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = 0.05 nm
2σ = -1.55 nm
α = 678.7 meV/V
score = 323 pt


sqd_gaafet_nanowire.msh2
168203 nodes, 32.0x10.0x9.0 nm

T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 0.50 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

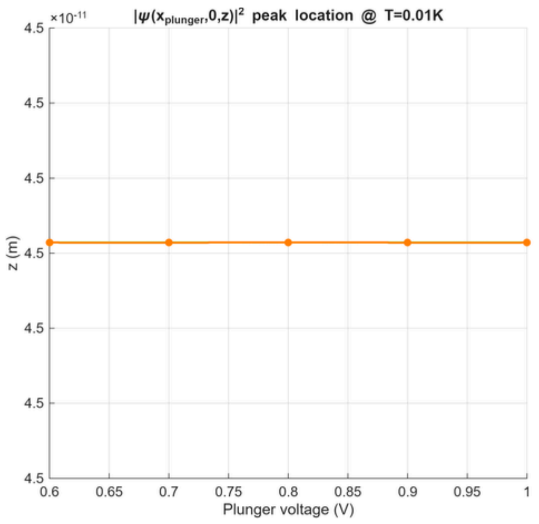
z_{dot} = 0.05 nm
2σ = -1.55 nm
α = 678.7 meV/V
score = 312 pt

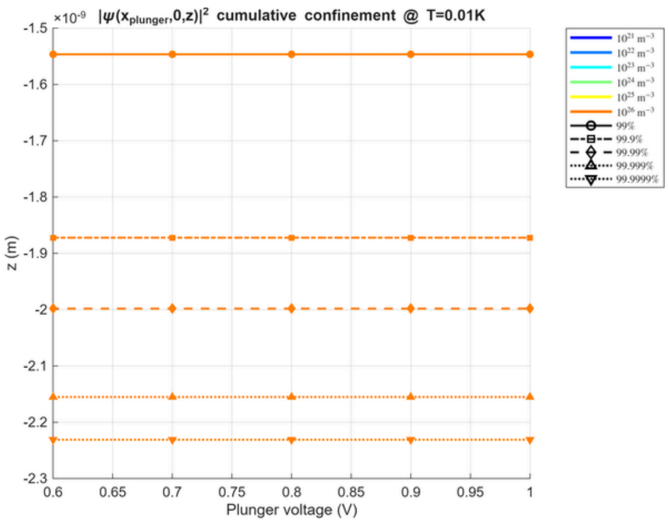

sqd_gaafet_nanowire.msh2
168203 nodes, 32.0x10.0x9.0 nm

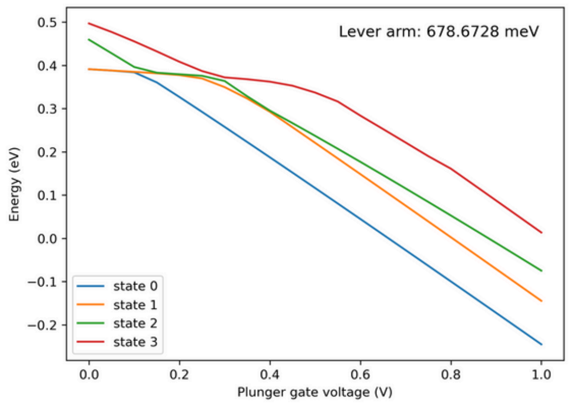
T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 1.20 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

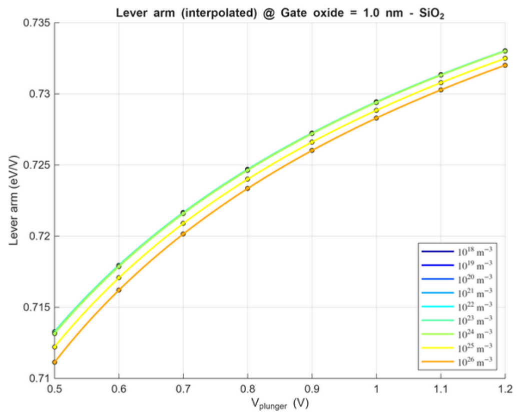
z_{dot} = 0.05 nm
2σ = -1.55 nm
α = 678.7 meV/V
score = 327 pt

Transport results

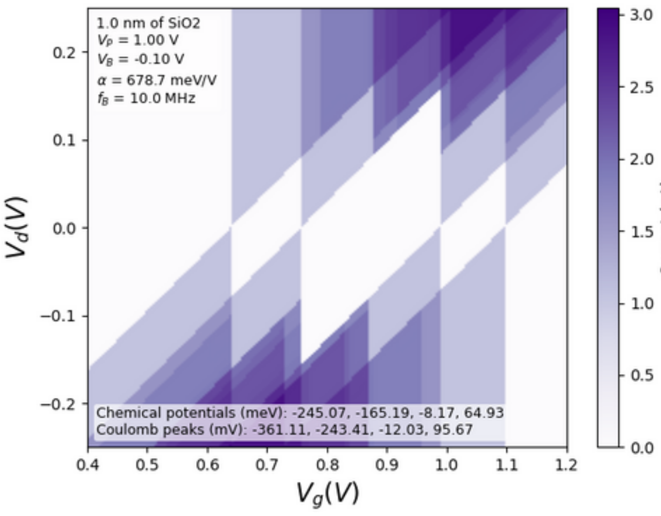






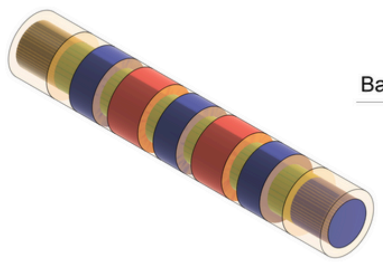


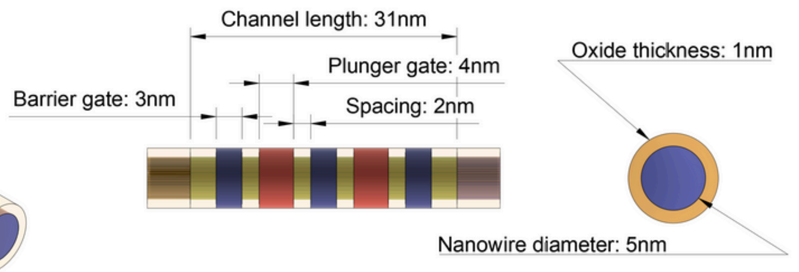
Charge Stability Diagram



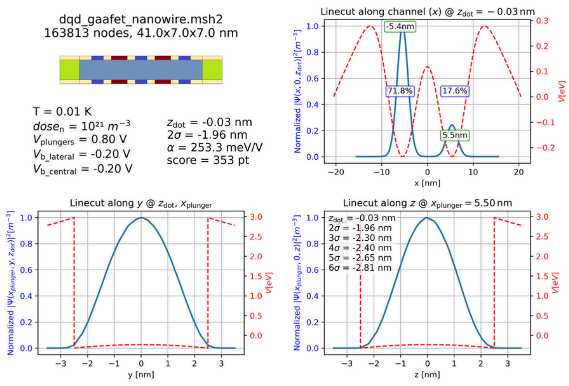
Structure 7
dqd_gaafet_nanowire

Double Quantum Dot
Gate All Around Nanowire
Only nanowire simulated





Confinement results

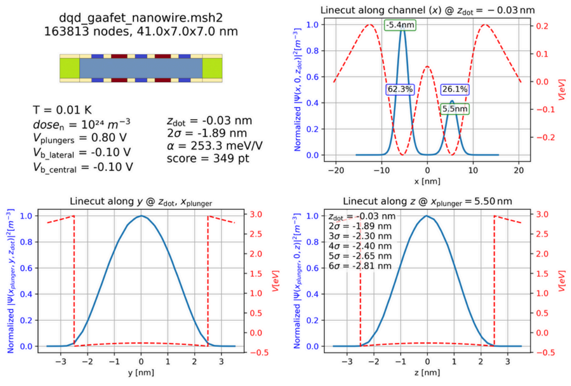


dqd_gaafet_nanowire.msh2
163813 nodes, 41.0x7.0x7.0 nm

T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 0.80 V
$V_{b\_lateral}$ = -0.20 V
$V_{b\_central}$ = -0.20 V

z_{dot} = -0.03 nm
2σ = -1.96 nm
α = 253.3 meV/V
score = 353 pt

z_{dot} = -0.03 nm
2σ = -1.96 nm
3σ = -2.30 nm
4σ = -2.40 nm
5σ = -2.65 nm
6σ = -2.81 nm

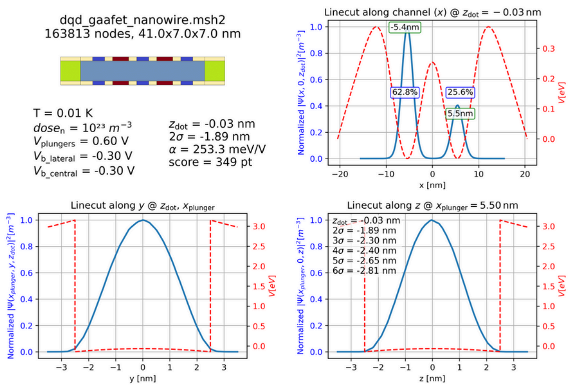


dqd_gaafet_nanowire.msh2
163813 nodes, 41.0x7.0x7.0 nm

T = 0.01 K
$dose_n = 10^{24}\ m^{-3}$
$V_{plungers}$ = 0.80 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

z_{dot} = -0.03 nm
2σ = -1.89 nm
α = 253.3 meV/V
score = 349 pt

z_{dot} = -0.03 nm
2σ = -1.89 nm
3σ = -2.30 nm
4σ = -2.40 nm
5σ = -2.65 nm
6σ = -2.81 nm

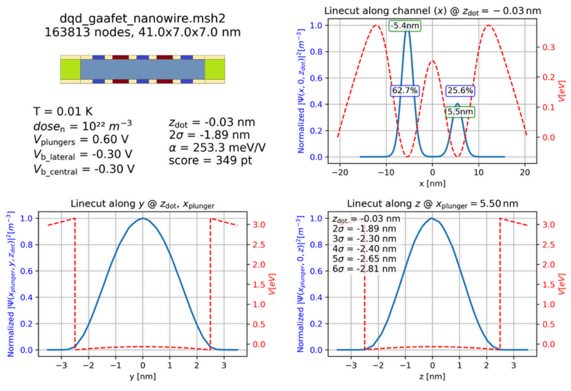


dqd_gaafet_nanowire.msh2
163813 nodes, 41.0x7.0x7.0 nm

T = 0.01 K
$dose_n = 10^{23}\ m^{-3}$
$V_{plungers}$ = 0.60 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = -0.03 nm
2σ = -1.89 nm
α = 253.3 meV/V
score = 349 pt

z_{dot} = -0.03 nm
2σ = -1.89 nm
3σ = -2.30 nm
4σ = -2.40 nm
5σ = -2.65 nm
6σ = -2.81 nm

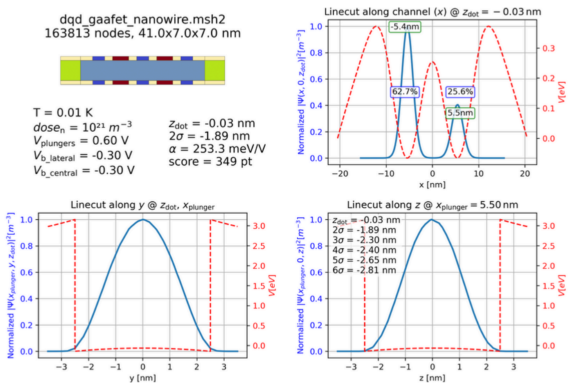


dqd_gaafet_nanowire.msh2
163813 nodes, 41.0x7.0x7.0 nm

T = 0.01 K
$dose_n = 10^{22}\ m^{-3}$
$V_{plungers}$ = 0.60 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = -0.03 nm
2σ = -1.89 nm
α = 253.3 meV/V
score = 349 pt

z_{dot} = -0.03 nm
2σ = -1.89 nm
3σ = -2.30 nm
4σ = -2.40 nm
5σ = -2.65 nm
6σ = -2.81 nm

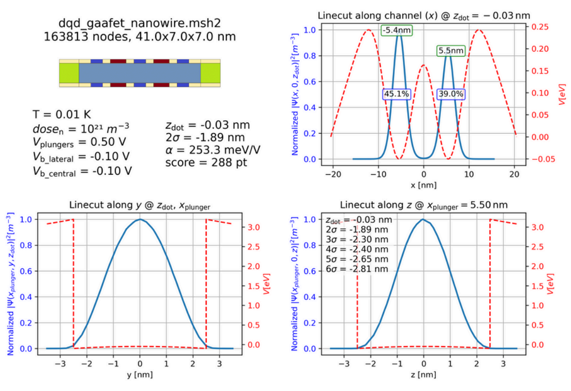


dqd_gaafet_nanowire.msh2
163813 nodes, 41.0x7.0x7.0 nm

T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 0.60 V
$V_{b\_lateral}$ = -0.30 V
$V_{b\_central}$ = -0.30 V

z_{dot} = -0.03 nm
2σ = -1.89 nm
α = 253.3 meV/V
score = 349 pt

z_{dot} = -0.03 nm
2σ = -1.89 nm
3σ = -2.30 nm
4σ = -2.40 nm
5σ = -2.65 nm
6σ = -2.81 nm



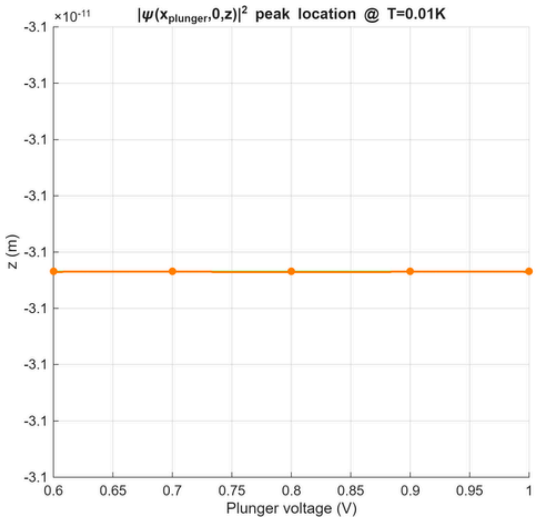
dqd_gaafet_nanowire.msh2
163813 nodes, 41.0x7.0x7.0 nm

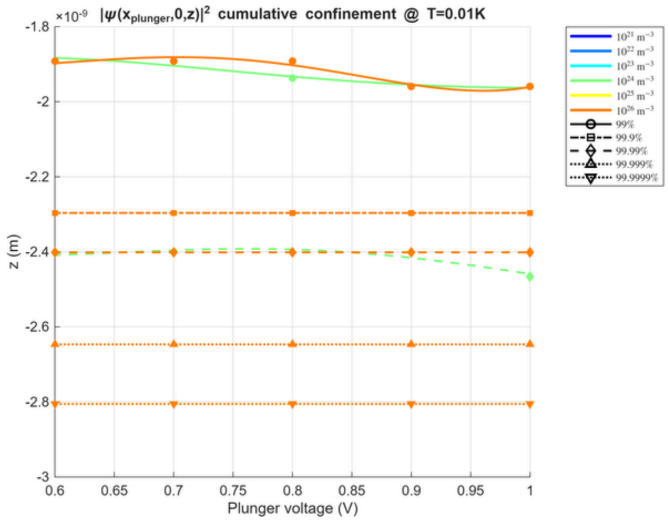
T = 0.01 K
$dose_n = 10^{21}\ m^{-3}$
$V_{plungers}$ = 0.50 V
$V_{b\_lateral}$ = -0.10 V
$V_{b\_central}$ = -0.10 V

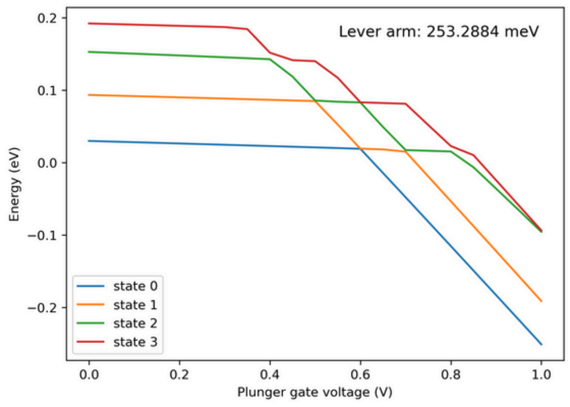
z_{dot} = -0.03 nm
2σ = -1.89 nm
α = 253.3 meV/V
score = 288 pt

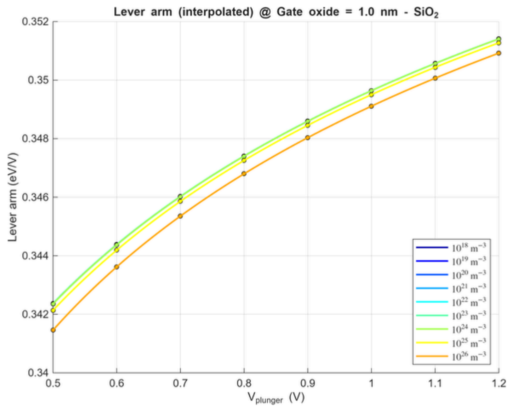
z_{dot} = -0.03 nm
2σ = -1.89 nm
3σ = -2.30 nm
4σ = -2.40 nm
5σ = -2.65 nm
6σ = -2.81 nm

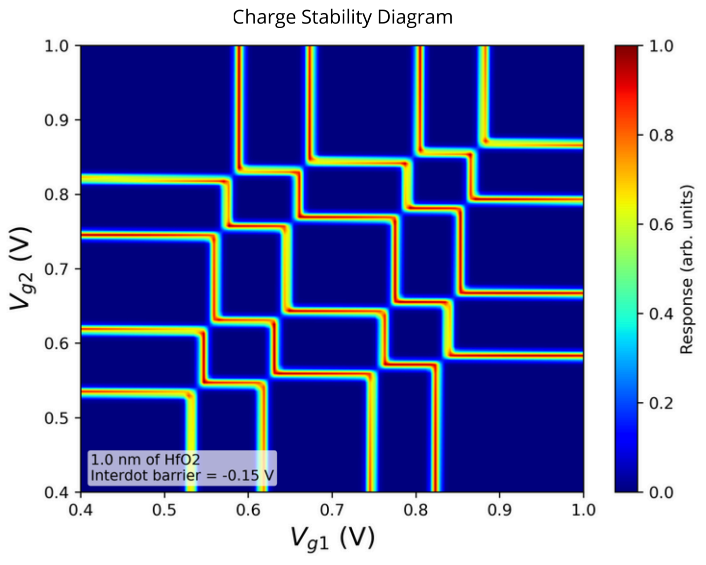
Transport results











Results Analysis and Comments

Overall, all the simulated devices exhibited satisfactory results that are consistent with expectations. The confinement analysis showed that all the devices can create a quantum dot in the intended location and it is possible to tune the wavefunction shape by modifying the input parameters. Charge stability diagrams show that, as expected, smaller structures are more effective at achieving confinement, as evidenced by the presence of larger Coulomb diamonds. However, such dimensions, in the order of only a few nanometers, are often extremely challenging to realize in practice. For this reason, larger devices, such as `sqd_soi_planar_dp` and `dqd_soi_planar_dp`, were also defined in order to study a complementary but scaled-up version of the system. The particle addition spectrum of the device `dqd_soi_planar`, which has a dimension of 43x14x7 nm, exhibits only diagonal lines. The lever arm matrix for this simulation indicates a nearly symmetric coupling of all gates to both dots. The values range from approximately 0.19 to 0.26 and the columns are almost identical, meaning that each gate influences both dots at the same time. This strong cross-capacitance explains why the stability diagrams displays diagonal lines rather than the expected rectangular or hexagonal cells. This limits independent control of the dots and the only apparent solution is to increase the device dimensions in order to further space the two plunger gates.

In DQD devices, it is common to observe asymmetric dot confinement, which can occur even when the geometry and gate voltages are nominally symmetric. Such asymmetries are partly due to numerical artifacts, where the discretization of the simulation domain and finite-element meshing introduce small energy offsets (on the order of a few μeV), which result in an unintended detuning between the two dots. [46] Tracy et al. (2010) demonstrate that this asymmetry arises in real devices from a combination of fabrication-related imperfections, such as variations in oxide thickness, local disorder in the semiconductor and trapped charges within the dielectric layers, all of which can locally modify the electrostatic potential. These effects may lead to differences in dot size, confinement strength or coupling to the gates, resulting in one dot being energetically higher or lower than the other. In experimental devices, such intrinsic asymmetries are often compensated by applying slightly different voltages to the plunger or barrier gates. This allows the potential landscape to be tuned, bringing the dots closer to energy degeneracy and enabling more symmetric confinement and controlled inter-dot tunnel coupling. [50]

The lever arm coefficients were computed for the first four states (ground state up to the third excited state) and exhibit some degree of nonlinearity. To better model the confinement simulations, a lever arm matrix was constructed and for any specific plunger voltage, an interpolation algorithm was used to retrieve the corresponding coefficient, even if it had not been computed directly. The lever arm coefficient can be reduced by modifying the capacitance of the associated gate, for instance by increasing the oxide thickness or using a low-k dielectric. A lower lever arm allows a wider range of plunger voltages, helping to mitigate instrumentation inaccuracy and noise. However, transport analysis shows that this also leads to less well-defined cells in the particle addition spectrum. Therefore, it is necessary to increase the lever arm to ensure suitable transport behaviour for DQD devices. To better model this phenomenon, a high-k dielectric, specifically hafnium dioxide (HfO_2), was employed for the computation of the particle addition spectrum to achieve more satisfactory results. In addition, the slopes of the transition lines in the charge stability diagram are determined by the lever arm matrix for a DQD device. Nearly orthogonal lines indicate that the plunger gates of one dot couple only weakly to the other, meaning cross-capacitance effects are negligible. As mentioned earlier, this behaviour can be attributed to the thinness of the gate oxide relative to the distance between the dots. For devices with thicker gate oxides, the transition lines would intersect at more obtuse angles. [31] Due to the fact that

$$C_G \sim \frac{\epsilon_r \epsilon_0 A}{t_{ox}}, \qquad \frac{C_{HfO_2}}{C_{SiO_2}} \approx \frac{25}{3.9} \approx 6.4$$

the use of hafnium dioxide resulted in transition lines that are 6.4 times less sloped, allowing the diagram to show the expected rectangular or hexagonal cell structure instead of diagonal lines. If the charge stability diagram shows lines artifacts as the plunger voltages increase, it may indicate that the dots are too strongly coupled. This issue can be mitigated by increasing the interdot barrier, by lowering the voltage applied to the central barrier gate. The plots that shows the ground state lever arm coefficient with respect to the applied plunger voltage for different values of doping concentration shows that for high enough voltages the dot energy slope tends to became constant and that it became independent of the doping concentration. In the ideal case, the lever arm coefficient should remain constant, meaning that the slope of the energy with respect to the plunger voltage would follow a linear relation independently of any other input parameter.

By analysing the plots that compare the variation of the probability density function peak position and its cumulative confinement levels with respect to the applied plunger voltage and the source/drain doping concentration, it becomes evident that higher voltages tend to pull the quantum dot closer to the gate oxide interface, thereby modifying its electrochemical potential through the lever arm coefficient. Given the fact that plunger voltages below $0.4 \div 0.5V$ often are not enough to guarantee a suitable local minima in the conduction band, thereby not leading to the formation of the quantum dot, it can be stated that all the planar devices exhibit a broadly similar behaviour, with a "pull ratio" of approximately $2nm/V$.

It is possible to observe that for gate all around nanowire devices, where the dot is localized at the center of the channel by the potential barrier given from the silicon-oxide interface, the study of z-axis peak location and cumulative confinement levels is not meaningful. In particular, the peak of the wavefunction is located at the exact same spot for all the plunger voltages, ideally at zero, but in practice exhibiting a negligible offset (on the order of 10^{-11}) due to numerical inaccuracies. This is consistent with expectations, as the use of a three-dimensional architecture provides improved electrostatic confinement due to the gates surrounding the entire channel.

In an old computation of the particle addition spectrum for the device `dqd_gaffet_nanowire` the expected pattern was partially interrupted by a central gap where charge transitions are absent. This effect can be attributed to few-electron regime that, due to the capacitive asymmetries, prevent both dots from being occupied simultaneously. These mechanisms lead to missing or broken charge transition lines, which manifest as a gap instead of closed stability cells. This issue was fixed by the use of a more refined mesh (from 163816 to 374905 nodes) for the charge stability diagram computation.

Chapter 9

Conclusions

In conclusion, the workflow proved to be effective and the customizable nature of the code makes it possible to adapt the simulation environment to more complex structures, while also enabling tailored post-processing of the obtained results. The simulation scripts were designed to easily accommodate modifications to further reduce human intervention, enabling the simulation of multiple devices and configurations in background while allowing attention to be focused on other tasks.

The simulated devices show behaviour that is in line with expectations. The confinement is guaranteed by the applied potential landscape and transport simulations shows that dot occupancies is possible with current and voltages magnitudes that are commonly found in the scientific literature. The only aspect that requires further investigation is the simulation of gate stack with different oxides, such as a thin film of silicon dioxide adhesion layer combined with a high-k dielectric. This would allow a more accurate evaluation of realistic gate capacitance and their impact on the quantum dots. The modelling of improved and more realistic charge stability diagrams is essential to determine the appropriate voltages needed to ensure correct carrier occupancies, as the current results shows that a thin film of silicon dioxide alone, for the implied device dimensions, is not sufficient to guarantee the expected cells on particle addition spectrums. Larger devices, or alternative architectures, may operate as intended when silicon dioxide is used, as reported in the literature. Furthermore, the use of more dense meshes could in practice provide high definition results and solve problems related to numerical inaccuracies that could lead to asymmetries and instability. It is necessary to avoid increasing the mesh node density excessively, as this could lead to convergence failures in the Poisson and Schrödinger solvers.

The fabrication simulation inside SProcess shows that the manufacturing of a planar device is feasible with standard state-of-the-art CMOS/VLSI processes. To more accurately model the fabrication process, a suitable damascene process should be implemented for the production of gate contacts. The one step that requires specific attention is the definition of the isotopically enriched silicon layer via epitaxial growth.

Chapter 10

Future Implementations

The development of the workflow for simulating custom CAD defined geometries in QT-CAD was time consuming due to the need to fix several issues and find workarounds for unsupported features. As a result, some of the originally planned ideas for this work were postponed, allowing more time and effort to be focused on the core objectives of the study. Some of these ideas could not be fully explored and are therefore considered for future implementations.

Primarily, once the core features for studying quantum confinement are refined, it becomes fundamentally important to extend the analysis to better simulate the device in a realistic scenario. Firstly, the focus was placed on the simulation of process variations. At the nanoscale, it becomes critical to study how these variations influence the quantum confinement behaviour of the device. Once the device can be reliably operated as a quantum dot, it can be simulated as a host for a spin qubit to study its dynamics and noise behaviour, with particular emphasis on identifying the physical parameters that influence the latter and how to improve the design.

Next, it may be valuable to simulate in QTCAD the results obtained from SProcess. In particular, extracting and reusing the resulting geometry, which provides a more realistic representation of how the device structure would appear after fabrication, could potentially lead to more accurate simulations results. Moreover, incorporating the extracted doping profiles, rather than approximating them as a discrete set of regions with uniform doping concentrations, could further enhance the reliability of the results.

10.1 Simulating Process Variations

It is of fundamental interest not only to simulate the behaviour of the device, but also how it will perform in the presence of process variations. At the nanoscale, the main process that may cause deviations from the simulated behaviour are those related to gate fabrication. To simulate variations in an automatic and systematically reproducible manner, without manually defining multiple variations of the same device geometry, a python script called `remesh.py` was employed, available in the attachments appendix as script C.2. The approach consists of merging all gate oxide volumes with the overall oxide, forming a single continuous volume. A new surface, with specific dimensions, is then positioned at a defined coordinate on the top of the oxide, labelled in Gmsh and used as the boundary for a gate contact. By repeating this procedure, all gate contacts can be created. Varying the gate surface dimensions in a controlled or random manner allows the simulation of process variations. Additional scripts can be developed to automatically define gate configurations, generate the corresponding mesh, run simulations and repeat this for a specified number of configurations, enabling analysis of how device behaviour changes with respect to gate variations, in a manner similar to the one employed by the batch runner, as described in Section 6.2.

The code begins by importing both the STEP and IGES files, with the filename specified as an argument to the script, e.g., `python remesh.py filename`. The Gmsh Open-Cascade kernel then imports all shapes (volumes in the CAD file) from the IGES file, which retains the names defined in the CAD software, into a new workspace. Each time an element is imported or modified, the workspace must be synchronized using `gmsh.model.occ.synchronize()`.
Exporting in IGES is not supported by the Gmsh kernel, so the output file will be exported as a STEP file.

```python
iges_file = f"{file_base}.iges"
step_file = f"{file_base}_mod.step"
output_file = "output.step"

if not os.path.exists(iges_file):
    print(f"Error: IGES file '{iges_file}' not found.")
    sys.exit(1)

gmsh.initialize()
gmsh.option.setNumber("General.Terminal", 1)
gmsh.model.add("modified_model")

gmsh.model.occ.importShapes(iges_file)
gmsh.model.occ.synchronize()
```

Next, the function `get_shapes_dict` retrieves all available shapes along with their tags and names. A for loop is then used to print them all.

```python
shapes_dict = get_shapes_dict()
print("[FOUND] Shapes found (name -> (dim, tag)):")
for name, ents in shapes_dict.items():
    print(f" - {name}: {ents}")
```

```
[FOUND] Shapes found (name -> (dim, tag)):
- drain_n:  [(3, 1)]
- drain_npp:  [(3, 2)]
- drain_np:  [(3, 3)]
- barrier_2:  [(3, 4)]
- plunger_2:  [(3, 5)]
- barrier_3:  [(3, 6)]
- oxide:  [(3, 7)]
- channel:  [(3, 8)]
- source_n:  [(3, 9)]
- source_npp:  [(3, 10)]
- source_np:  [(3, 11)]
- plunger_1:  [(3, 12)]
- barrier_1:  [(3, 13)]
```

Once the software has assigned a tag to each shape, the function `union_shapes_by_labels` can be used to merge all oxide regions into a single volume. The tag of the resulting oxide shape is then retrieved by scanning all shapes.

```python
labels_to_merge = ["plunger_1", "plunger_2", "barrier_1",
 ↪  "barrier_2", "barrier_3", "oxide"]
merged = union_shapes_by_labels(labels_to_merge, shapes_dict)
resulting_volumes = gmsh.model.occ.getEntities(dim=3)
oxide_tag = 0
for dim, tag in resulting_volumes:
    name = gmsh.model.getEntityName(dim, tag)
    if name == "Shapes/oxide":
        oxide_tag = tag
```

Next, the function `get_z_max_of_label` retrieves the maximum z-axis point value of the shape with the tag `oxide`. The value is rounded to avoid numerical inaccuracies. Then, the function `add_surface` inserts a new surface at the previously determined z coordinate. In this example, the x and y coordinates of the surface center are both 0 and the gate surface has dimensions 20×10 (arbitrary scale in Gmsh, nanometers in QTCAD) with the name `top_contact`. The function automatically returns its tag value.

```
z = round(get_z_max_of_label("oxide", shapes_dict), 1)
surf_tag = add_surface(0, 0, 20, 10, z, name="top_contact")
print(f"[DONE] Surface 'top_contact' added with tag {surf_tag}")
```

Next, the new gate surface must be merged with the top oxide surface. To do this, the program requires the surface tag, which is not known in advance. A custom function, `print_surface_tags_with_info`, is used to print all surface tags along with their relevant information.

```
print_surface_tags_with_info("Shapes/oxide")
```

```
[SEARCH] Surfaces of volume 'Shapes/oxide' (tag=12):
- Surface tag:  94 | center = [0.0, 0.0, 2.0]
- Surface tag:  95 | center = [0.0, -7.0, -8.000000000000002]
- Surface tag:  96 | center = [35.2, 0.0, -8.000000000000002]
- Surface tag:  97 | center = [0.0, 7.0, -8.000000000000002]
- Surface tag:  98 | center = [-35.2, 0.0, -8.000000000000002]
- Surface tag:  99 | center = [-33.2, 0.0, -7.000000000000002]
- Surface tag:  100 | center = [0.0, 5.0, -7.000000000000002]
- Surface tag:  101 | center = [-21.5, 0.0, 0.9999999999999998]
- Surface tag:  102 | center = [0.0, -5.0, -7.000000000000002]
- Surface tag:  103 | center = [33.2, 0.0, -7.000000000000002]
- Surface tag:  104 | center = [21.5, 0.0, 0.9999999999999998]
- Surface tag:  105 | center = [0.0, 0.0, -18.0]
- Surface tag:  106 | center = [0.0, 0.0, -16.0]
- Surface tag:  107 | center = [0.0, 0.0, 0.0]
```

These tag values do not correspond directly to the surface numbers displayed in Gmsh, so a trial-and-error approach is required to manually identify the surface tag corresponding to the top oxide surface. This tag is then used in the following lines of code to merge the newly added gate surface with the top oxide surface via the `gmsh.model.occ.fragment` command. This function expects a list of tuples, where each tuple represents a single element. In the tuple (2, surface_tag), the first number indicates the element's dimension, 2 in this case, corresponding to a surface, while the second is the surface tag.

```
top_tag = 94    # To be found manually

fused_surfaces, _ = gmsh.model.occ.fragment([(2, surf_tag)], [(2,
↪  top_tag)])
gmsh.model.occ.synchronize()
```

At this point, the oxide shape has three distinct top surfaces: the original surface, the newly added gate surface and the fused surface. The first two must be removed so that only the fused surface remains. Then the output file is exported.

```
gmsh.model.occ.remove([(2, surf_tag)])
gmsh.model.occ.synchronize()

gmsh.model.occ.remove([(2, top_tag)])
gmsh.model.occ.synchronize()

gmsh.write(output_file)
gmsh.finalize()
```

This step contains the main issue that prevents the script from working correctly and needs to be addressed. The main problem is that the new surface fails to merge with the rest of the shape, preventing proper assignment to physical groups in Gmsh and, consequently, the quantum simulation in QTCAD. A thorough understanding of how the OpenCascade kernel operates is required to address this issue and correctly define all gate surfaces, enabling the proper introduction of process variations into the mesh and simulation environment. The final mesh, as well as an example of meshing issue, are shown in figure 10.1.

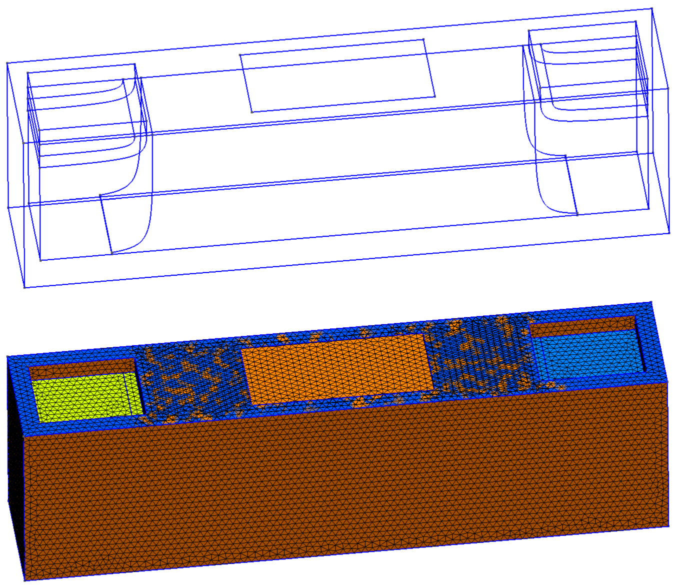


Figure 10.1: Gate definition in Gmsh for process variations

10.2 Mesh Extraction From Sentaurus Process

In Sentaurus Process, the resulting geometry provides a close approximation of the device as it would appear following actual fabrication steps. Some additional steps can be performed to extract the final geometry, apply post-processing cleaning and adaptations and then prepare it for handling in Gmsh and simulation in QTCAD. Analysing the quantum behaviour of the actual device, as fabricated in Sentaurus, provides an additional level of fidelity, allowing a better understanding of device performance in a realistic context rather than assuming a perfect geometry. As of now, there is no conclusive evidence that this concept can be realized. First, the mesh needs to be extracted from Sentaurus and then preprocessed to be handled in Gmsh. A thorough understanding of how the software applies physical groups is required. In this work, physical group assignments were performed on the geometries only. Assigning them directly on meshes is non-trivial and requires custom workflows and several workarounds, as this functionality is not natively supported by Gmsh and compatibility with QTCAD is even less straightforward. Addressing this issue could be a subject of future work to better visualize the effects of a realistic structure on quantum behaviour. However, modelling a device using the Sentaurus mesh as a reference currently remains the easiest and fastest way to achieve reliable results.

10.3 Simulating Qubit Package

This work focused on the study of quantum confinement. The next step is to simulate the device as a qubit. In cases where simulations showed that the device correctly formed a quantum dot at the desired location, it is of fundamental interest to consider that dot as a potential site for a spin qubit. QTCAD provides a versatile framework for modelling and simulating spin qubits, with particular emphasis on Electric Dipole Spin Resonance (EDSR). In EDSR, a time-dependent voltage bias is applied to manipulate the spin of an electron confined in a quantum dot. This manipulation is enabled by spin-orbit coupling, which may either be intrinsic to the material or induced artificially via a micromagnet. After defining the quantum dot, a magnetic field is applied to introduce the Zeeman splitting, which separates the spin states. Then, a time-dependent voltage is applied to one of the gates to actively manipulate the spin of the electron. Finally, the device dynamics are simulated, allowing the study of phenomena such as Rabi oscillations and the coherent evolution of the spin qubit.

Part III

Attachments

Appendix A

Python Tool Scripts

A.1 device_config.py

```python
import re
import sys

if len(sys.argv) < 2:
    sys.exit(1)

input_file = sys.argv[1]
dev_name = "d"
output_file = "device_config.txt"

volume_pattern = r'Physical Volume\("([^"]+)"\)\s*=\s*\{[^}]*\};'
surface_pattern = r'Physical Surface\("([^"]+)"\)\s*=\s*\{[^}]*\};'

with open(input_file, "r") as f:
    content = f.read()
    volumes = re.findall(volume_pattern, content)
    surfaces = re.findall(surface_pattern, content)

with open(output_file, "w") as f:
    f.write('''# MATERIAL PARAMETERS\n
semiconductor = mt.Si
p_doping = 0*1e6
n_doping = 0*1e6
dielectric = mt.SiO2
metal_workfunction= semiconductor.chi+ semiconductor.Eg/2
\n
''')

f.write("# REGION AND BOUNDARIES CONDITION DEFINITION\n\n")

```

```python
# VOLUMES
volume_lines = []
for name in volumes:
    if name.startswith("semi_"):
        args = "semiconductor"
    elif name.startswith("nsemi_"):
        args = "semiconductor, pdoping=0, ndoping=n_doping"
    elif name.startswith("psemi_"):
        args = "semiconductor, pdoping=p_doping, ndoping=0"
    elif name.startswith("diel_"):
        args = "dielectric"
    else:
        continue
    volume_lines.append((name, args))

if volume_lines:
    max_len = max(len(name) for name, _ in volume_lines)
    for name, args in volume_lines:
        spaces = " " * (max_len - len(name))
        f.write(f'{dev_name}.new_region("{name}",{spaces} {args})\n')

f.write("\n")

# SURFACES
surface_lines = []
for name in surfaces:
    if name.startswith("gatebnd_"):
        surface_lines.append((name, f"{dev_name}.new_gate_bnd(\"{name}\", 0, metal_workfunction)"))
    elif name.startswith("ohmicbnd_"):
        surface_lines.append((name, f"{dev_name}.new_ohmic_bnd(\"{name}\")"))

if surface_lines:
    for line in surface_lines:
        f.write(line[1] + "\n")
```

Listing A.1: `device_config.py` - Generates a QTCAD API python code for the definition of volumes and surfaces physical groups, starting from the GMSH `.geo` file where the mesh is defined, to do a copy&paste inside the simulation script. This is used to save time and the use of this script is absolutely optional.

A.2 `mesh_volume.py`

```python
import meshio
import numpy as np

def tet_volume(p0, p1, p2, p3):
    # Computes the volume of a single tetrahedron
    return np.abs(np.dot((p1 - p0), np.cross(p2 - p0, p3 - p0))) / 6.0

def volume(mesh_path):
    # Computes the total volume of a tetrahedral mesh
    # Returns None in case of an error
    try:
        mesh = meshio.read(mesh_path, file_format="gmsh")
        points = mesh.points
        tetra_cells = mesh.cells_dict.get("tetra", None)

        if tetra_cells is None:
            raise ValueError("The mesh does not contain tetrahedral elements.")

        total_volume = 0.0
        for tet in tetra_cells:
            p0, p1, p2, p3 = points[tet]
            total_volume += tet_volume(p0, p1, p2, p3)

        return total_volume
    except Exception as e:
        print(f"[ERROR] Volume calculation failed: {e}")
        return None
```

Listing A.2: `mesh_volume.py` - Starting from the mesh file path, this script calculates the volume of the latter. This is used to define the desired number of nodes the mesh should have based on the granularity factor (such as nodes over cubic nanometer). This parameter can be used in the adaptive Poisson solver in the simulation file. Note: even if this script can help automate the simulation, it takes some time and may represent a huge time sink when dealing with batch simulations. Also, with the import of CAD-defined geometries, the adaptive poisson solver does not work right now. A better way is to look at the geometry volume in the CAD software or by using some online tools such as `https://3dviewer.net` and then define by hand the desired granularity and tweak the `Mesh.MeshSizeMax` parameter inside GMSH.

A.3 `z_for_threshold.py`

```python
import numpy as np

def z_for_threshold(z_nm, pdf, threshold=0.99):
    # Sort z from positive to negative
    sort_idx = np.argsort(-z_nm)
    z_sorted = z_nm[sort_idx]
    pdf_sorted = pdf[sort_idx]

    # Calculate dz (with positive sign)
    dz = np.abs(np.gradient(z_sorted))

    # Calculate total integral for normalization
    total_area = np.sum(pdf_sorted * dz)
    if total_area <= 0:
        raise ValueError("Total integral is zero or negative
           invalid pdf.")

    # Calculate normalized cumulative sum
    cumulative = np.cumsum(pdf_sorted * dz) / total_area

    # Find first index where cumulative exceeds the threshold
    if cumulative[-1] < threshold:
        idx = len(z_sorted) - 1
    else:
        idx = np.searchsorted(cumulative, threshold)

    return z_sorted[min(idx, len(z_sorted)-1)]
```

Listing A.3: `z_for_threshold.py` - This function takes as input the z coordinate in nanometers, the normalized probability density function along that axis and a threshold factor. It calculates where the integral of the normalized probability density function starting from the beginning of the z axis reach the threshold parameter. This is used to compute the probability density function z-axis cumulative confinement levels, in order to see at which depth the carrier is confined with a certain fidelity.

A.4 mat_merge.py

```python
import os
import numpy as np
from scipy.io import loadmat, savemat

def merge_dot_results(base_folder="batch_results", output_file="dot_z_locations.mat"):

    folders = [os.path.join(base_folder, d) for d in os.listdir(base_folder) if os.path.isdir(os.path.join(base_folder, d))]

    V_plunger_list = []
    n_doping_list = []
    dot_z_list = []
    th_2N_list = []
    th_3N_list = []
    th_4N_list = []
    th_5N_list = []
    th_6N_list = []

    for folder in folders:
        mat_path = os.path.join(folder, "dot_results.mat")
        if os.path.exists(mat_path):
            data = loadmat(mat_path)
            V_plunger_list.append(float(data.get("V_plunger", [np.nan])[0]))
            n_doping_list.append(float(data.get("doping", [np.nan])[0]))
            dot_z_list.append(float(data.get("dot_z", [np.nan])[0]))
            th_2N_list.append(float(data.get("dot_z_threshold_2N", [np.nan])[0]))
            th_3N_list.append(float(data.get("dot_z_threshold_3N", [np.nan])[0]))
            th_4N_list.append(float(data.get("dot_z_threshold_4N", [np.nan])[0]))
            th_5N_list.append(float(data.get("dot_z_threshold_5N", [np.nan])[0]))
            th_6N_list.append(float(data.get("dot_z_threshold_6N", [np.nan])[0]))
        else:
            print(f"Warning: cannot find {mat_path}!")

    # Convert in numpy array
    V_plunger_arr = np.array(V_plunger_list)
    n_doping_arr = np.array(n_doping_list)
    dot_z_arr = np.array(dot_z_list)
    th_2N_arr = np.array(th_2N_list)
    th_3N_arr = np.array(th_3N_list)
    th_4N_arr = np.array(th_4N_list)
    th_5N_arr = np.array(th_5N_list)
    th_6N_arr = np.array(th_6N_list)

    # Save merged file
    savemat(output_file, {
        "V_plunger": V_plunger_arr,
        "n_doping": n_doping_arr,
        "dot_z": dot_z_arr,
        "threshold_2N": th_2N_arr,
        "threshold_3N": th_3N_arr,
        "threshold_4N": th_4N_arr,
        "threshold_5N": th_5N_arr,
        "threshold_6N": th_6N_arr
    })

    print(f"File '{output_file}' successfully created.")
```

Listing A.4: `mat_merge.py` - This function is used at the end of a batch simulation. It takes all of the `dot_results.mat` files and merge the results about the vertical confinement into one `.mat` file. This can then be fed to a Matlab script in order to plot the batch results and look at how the probability density function z-axis peak location and cumulative confinement levels behaves with respect to the plunger voltage and contacts doping concentration.

A.5 dot_range.py

```python
def is_within_percentage(value, reference, percent):
    margin = reference * percent / 100
    return (reference - margin) <= value <= (reference + margin)
```

Listing A.5: `dot_range.py` - This simple function return a boolean value, checking if two values are close with a deviation threshold percentage.

A.6 x_for_threshold.py

```python
import numpy as np

def crop_around_coordinate(pos, psi2, center, width):
    # Crop data around a coordinate
    half_width = width / 2
    mask = (pos >= center - half_width) & (pos <= center +
        half_width)
    return pos[mask], psi2[mask]

def x_for_threshold(psiposx, psix0, plunger_x_coordinate,
   crop_width_x):
    # Cropping the region around the plunger
    x_crop, psi_x_crop = crop_around_coordinate(psiposx, psix0,
        plunger_x_coordinate, crop_width_x)

    # Integrals using the trapezoidal rule
    total_integral = np.trapz(psix0, psiposx)
    cropped_integral = np.trapz(psi_x_crop, x_crop)

    # Avoid division by zero
    if total_integral == 0:
        return 0.0

    # Return the confinement percentage
    return cropped_integral / total_integral
```

Listing A.6: `x_for_threshold.py` - This script includes two functions. `crop_around_coordinate` takes a x and y array couple and coordinates where to crop the array. This can be used on its own or in combination with the second function, `x_for_threshold`, used to determine with integration the area of the wavefunction under the plunger gate in relation with the total area. This is useful to get information about the confinement goodness.

A.7 apex.py

```python
def get_apex(number, digits=2):
    if number == 0:
        return "0"

    # Extract mantissa and exponent
    format_str = f"{{:.{digits}e}}"
    mantissa_str, exponent_str = format_str.format(number).split("e")
    mantissa = float(mantissa_str)
    exponent = int(exponent_str)

    # Convert to Unicode superscript
    superscript_map = str.maketrans("0123456789-", "")
    superscript_exp = str(exponent).translate(superscript_map)

    if mantissa == 1.0:
        return f"10{superscript_exp}"
    else:
        return f"{mantissa} 10 {superscript_exp}"
```

Listing A.7: `apex.py` - This script returns a scientific notation for displaying a variable as, for example, 10^{22} and not $1e + 22$. Used for the doping concentration display in the output images.

A.8 get_exported_parameters.py

```python
import csv

def get_parameter_value(csv_path, target_name):
    with open(csv_path, newline='', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        next(reader)

        for row in reader:
            if row[0].strip() == target_name:
                try:
                    value = float(row[3])
                    return value
                except ValueError:
                    raise ValueError(f"Value '{target_name}'
                        cannot be casted into float: {row[3]}")

        raise KeyError(f"Cannot find parameter '{target_name}'.")
```

Listing A.8: `get_exported_parameters.py` - This function takes as input the path of a CSV file, formatted as Fusion does, and a parameter label. It returns the corresponding value, parsed into float.

A.9 `remove_fails.py`

```python
import os
from pathlib import Path

def clean_batch_results():
    root = Path.cwd() / "batch_results"
    if not root.exists() or not root.is_dir():
        print("[ERROR] Cannot find directory 'batch_results'.")
        return

    print(f"[INFO] Analyzing: {root.resolve()}")

    # Cerca ricorsivamente
    for file in root.rglob("*.png"):
        filename = file.name

        if filename.startswith("X_"):
            try:
                file.unlink()
                print(f"[DELETED] {file}")
            except Exception as e:
                print(f"[ERROR] Cannot delete {file}: {e}")

        elif filename.startswith("Y_"):
            new_name = filename[2:]  # Remove "Y_"
            new_path = file.with_name(new_name)

            try:
                file.rename(new_path)
                print(f"[RENAMED] {file.name} -> {new_name}")
            except Exception as e:
                print(f"[ERROR] Cannot rename {file}: {e}")

if __name__ == "__main__":
    clean_batch_results()
```

Listing A.9: `remove_fails.py` - This script is used to delete all the `.png` files contained in a specific directory that starts with X$.Itisusedtoautomaticallydeletethenon-workingconfigurationsgeneratedbythebatchrunner.$

A.10 compute_leverarm.py

```python
import os
import scipy.io as sio
import numpy as np
import sys
sys.path.append('/opt/qtcad-1.4.3/qtcad/')

from qtcad.device import constants as ct
from qtcad.device import Device
from qtcad.device.leverarm import Solver as LeverArmSolver
from qtcad.device.leverarm import SolverParams as
    LeverArmSolverParams
from scipy.io import savemat, loadmat
from qtcad.device.poisson import SolverParams as
    PoissonSolverParams
from qtcad.device.schrodinger import SolverParams as
    SchrodingerSolverParams

import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt

def compute_single_lever_arm(device, boundary, bias):
    params_poisson = PoissonSolverParams()
    params_poisson.tol = 1e-5
    params_schrod = SchrodingerSolverParams()
    params_schrod.num_states = 10

    lever_arm_solver_params = LeverArmSolverParams({
        "pot_solver_params": params_poisson,
        "schrod_solver_params": params_schrod
    })

    # Create and run the lever arm solver
    print("Getting lever arm data for plunger gate")
    plunger_voltages = [bias-0.05, bias, bias+0.05]
    lever_arm_slv = LeverArmSolver(device, [boundary],
        plunger_voltages,
                                   dot_region=device.dot_region,
                                       solver_params=
                                       lever_arm_solver_params)
    poly_coeffs = lever_arm_slv.solve()
    return np.abs(poly_coeffs[0]) / ct.e

def compute_lever_arm(device, boundary):
    # Create the parameters of the lever-arm solver
    params_poisson = PoissonSolverParams()
    params_poisson.tol = 1e-5
    params_schrod = SchrodingerSolverParams()
    params_schrod.num_states = 10

    lever_arm_solver_params = LeverArmSolverParams({
        "pot_solver_params": params_poisson,
        "schrod_solver_params": params_schrod
    })

    # Create and run the lever arm solver
    print("Getting lever arm data for plunger gate")
    plunger_voltages = np.linspace(0, 1, 21)
    lever_arm_slv = LeverArmSolver(device, [boundary],
       plunger_voltages,
                                   dot_region=device.dot_region,
                                      solver_params=
                                      lever_arm_solver_params)
    poly_coeffs = lever_arm_slv.solve()

    lever_arm_value = np.abs(poly_coeffs[0]) / ct.e
    print(f"The lever arm is {lever_arm_value}")

    # Plot results
    fig, ax1 = plt.subplots()

    ax1.set_ylabel("Energy (eV)")
    ax1.set_xlabel("Plunger gate voltage (V)")

    # Dictionary to be saved in .mat
    mat_data = {"lever_arm": lever_arm_value}

    for i, data in enumerate(lever_arm_slv.energies.T / ct.e):
        ax1.plot(plunger_voltages, data, label=f"state {i}")
        # Save bias and energy for each state
        mat_data[f"bias_{i}"] = plunger_voltages
        mat_data[f"energy_{i}"] = data

    # Set figure size and DPI
    fig.set_size_inches(7, 5)
    fig.set_dpi(300)

    # Add legend in the lower left corner
    plt.legend(loc='lower left')

    text_str = f"Lever arm: {lever_arm_value*1000:.4f} meV"

    # Add text in the top right corner inside the plot area
    ax1.text(0.95, 0.95, text_str,
             horizontalalignment='right',
                verticalalignment='top',
                transform=ax1.transAxes,
                fontsize=12)

    plt.savefig("lever_arm.png", dpi=300, bbox_inches='tight')

    # Save the dictionary into a .mat file
    savemat("lever_arm.mat", mat_data)
```

Listing A.10: `compute_leverarm.py` - This script is used to calculate the leverarm coefficients of a generic device, passed as argument. It can calculate a single value, used for defining the electrochemical potential inside a dot starting from a gate voltage, or, by doing a sweep, calculate rthe coefficients for different values of potential and different number of states. This is done to generate an illustrative summary.

A.11 get_leverarm.py

```python
import os
import scipy.io as sio
import numpy as np
import sys
import inspect
sys.path.append('/opt/qtcad-1.4.3/qtcad/')
from qtcad.device import Device
from compute_leverarm import compute_lever_arm

def get_parent_mat_path():
    """
    Returns the path of lever_arm.mat located in the parent directory
    of the current script.
    """
    script_path = os.path.abspath(inspect.getfile(inspect.currentframe()))
    parent_dir = os.path.abspath(os.path.join(os.path.dirname(script_path), '..'))
    return os.path.join(parent_dir, 'lever_arm.mat')


def get_lever_arm(device, boundary):
    """
    Reads and returns the single lever arm value stored in lever_arm.mat
    located in the parent directory. If the file does not exist, it calls
    compute_lever_arm() to create it.
    """
    mat_file_path = get_parent_mat_path()

    if not os.path.isfile(mat_file_path):
        print("lever_arm.mat not found. Creating it by running compute_lever_arm()...")
        compute_lever_arm(device, boundary)

    mat_contents = sio.loadmat(mat_file_path)
    if 'lever_arm' not in mat_contents:
        raise KeyError(f"'lever_arm' key not found in {mat_file_path}")

    lever_arm_value = float(mat_contents['lever_arm'].squeeze())
    print(f"Lever arm value read from file: {lever_arm_value}")
    return lever_arm_value

```

```python

def energy_to_bias(device, boundary, state, energy):
    """
    Given a state (int) and an energy (float), returns the bias voltage
    corresponding to the energy nearest to the given one in the data.

    If lever_arm.mat does not exist, calls compute_lever_arm(device).
    """
    mat_file_path = get_parent_mat_path()

    if not os.path.isfile(mat_file_path):
        print(f"{mat_file_path} not found. Creating it by running compute_lever_arm()...")
        compute_lever_arm(device, boundary)

    mat_contents = sio.loadmat(mat_file_path)

    bias_key = f"bias_{state}"
    energy_key = f"energy_{state}"

    if bias_key not in mat_contents or energy_key not in mat_contents:
        raise KeyError(f"Keys '{bias_key}' or '{energy_key}' not found in {mat_file_path}")

    bias_array = mat_contents[bias_key].squeeze()
    energy_array = mat_contents[energy_key].squeeze()

    # Find index of nearest energy value
    idx = np.abs(energy_array - energy).argmin()
    return bias_array[idx]


def bias_to_energy(device, boundary, state, bias):
    """
    Given a state (int) and a bias voltage (float), returns the energy
    corresponding to the bias nearest to the given one in the data.

    If lever_arm.mat does not exist, calls compute_lever_arm(device).
    """
    mat_file_path = get_parent_mat_path()

    if not os.path.isfile(mat_file_path):
        print(f"{mat_file_path} not found. Creating it by running
            compute_lever_arm()...")
        compute_lever_arm(device, boundary)

    mat_contents = sio.loadmat(mat_file_path)

    bias_key = f"bias_{state}"
    energy_key = f"energy_{state}"

    if bias_key not in mat_contents or energy_key not in
       mat_contents:
        raise KeyError(f"Keys '{bias_key}' or '{energy_key}' not
            found in {mat_file_path}")

    bias_array = mat_contents[bias_key].squeeze()
    energy_array = mat_contents[energy_key].squeeze()

    # Find index of nearest bias value
    idx = np.abs(bias_array - bias).argmin()
    return energy_array[idx]
```

Listing A.11: `get_leverarm.py` - This script handles the store and extraction of leverarm coefficientrs inside the `lever_arm.mat` file. It the file does not exists, it use the functions inside `compute`$_{l}everarm.pytocalculatethem.$

Appendix B

Python Simulation Scripts

B.1 sim_dqd.py

```python
import os
import sys
sys.path.append('/opt/qtcad-1.4.3/qtcad/')

from qtcad.device import constants as ct
from qtcad.device.mesh3d import Mesh, SubMesh
from qtcad.device.analysis import linecut
from qtcad.device import io
from qtcad.device import analysis
from qtcad.device import materials as mt
from qtcad.device import Device, SubDevice
from qtcad.device.poisson import Solver as PoissonSolver
from qtcad.device.poisson import SolverParams as
    PoissonSolverParams
from qtcad.device.schrodinger import Solver as SchrodingerSolver
from qtcad.device.schrodinger import SolverParams as
    SchrodingerSolverParams
from qtcad.device.leverarm import Solver as LeverArmSolver
from qtcad.device.leverarm import SolverParams as
    LeverArmSolverParams
from scipy.io import savemat, loadmat
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
import matplotlib.image as mpimg
import pathlib
from pathlib import Path
import numpy as np
import math

```

```
# IMPORT CUSTOM FUNCTIONS
sys.path.append(os.path.join(os.path.dirname(__file__), "tools"))
from z_for_threshold import z_for_threshold
from dot_range import is_within_percentage
from get_leverarm import get_lever_arm
from get_leverarm import energy_to_bias
from get_leverarm import bias_to_energy
from dot_area import area_vista_plunger
from dot_area import area_vista_contacts
from get_exported_parameters import get_parameter_value
from compute_leverarm import compute_single_lever_arm
from charge_stability_diagram import
    compute_charge_stability_diagram
from x_for_threshold import x_for_threshold,
    crop_around_coordinate
from apex import get_apex



# -------------------------------------------------------------- #
# DEFINE PATHS TO INPUT AND OUTPUT FILES                         #
# -------------------------------------------------------------- #

script_dir = pathlib.Path(__file__).parent.resolve()

# DEFINE DEVICE NAME, USED TO FIND INPUT FILES
device_name = "dqd_compact"

mesh_name   = device_name + ".msh2"
image_name  = device_name + ".png"
parameters_name = device_name + ".csv"

config_dir = script_dir / "config"

path_mesh   = config_dir / "meshes" / mesh_name
path_image  = config_dir / "images" / image_name
path_parameters = config_dir / "exported_parameters" /
    parameters_name

results_dir = script_dir / "results"

path_hdf5 = results_dir / "device_results.hdf5"
path_psi0 = results_dir / "device_psi0.vtu"
path_psi1 = results_dir / "device_psi1.vtu"

path_results_mat = results_dir / "device_results.mat"
path_results_img = results_dir / "device_results.png"


```

```python


# ------------------------------------------------------------- #
# DEVICE PARAMETERS                                              #
# ------------------------------------------------------------- #

# IMMEDIATELY FIND OUT IF BATCH RUNNING IS ENABLED
batch = os.getenv("BATCH", "False")

# CHECK IF WE ARE IN BATCHING AND UPDATE SOME PATHS
if batch == "True":
    path_parameters = Path("../../config/exported_parameters") / parameters_name
    path_hdf5 = "device_results.hdf5"
    path_results_mat = "device_results.mat"

# EXTRACT THE GEOMETRICAL DIMENSIONS FROM THE .CSV FILE EXPORTED FROM FUSION360
gate_oxide      = get_parameter_value(path_parameters, "gate_oxide_thickness"   )*1e-9
lateral_oxide   = get_parameter_value(path_parameters, "oxide_lateral"      )*1e-9
plunger_length  = get_parameter_value(path_parameters, "plunger_gate"       )*1e-9
barrier_length  = get_parameter_value(path_parameters, "barrier_gate"       )*1e-9
plunger_width   = get_parameter_value(path_parameters, "channel_width"      )*1e-9
spacers         = get_parameter_value(path_parameters, "spacers"            )*1e-9
contact_spacers = get_parameter_value(path_parameters, "contact_spacers"    )*1e-9

# COMPUTE SOME DIMENSIONS FROM EXTRACTED PARAMETERS
contact_dot_spacing = spacers + contact_spacers + barrier_length + plunger_length/2
opposite_contact_dot_spacing = 3*spacers + contact_spacers + 2*barrier_length + plunger_length*3/2
dot_spacing = barrier_length + 2*spacers + plunger_length
plunger_x_coordinate = (barrier_length + plunger_length)/2 + spacers

# DEFINE THE DEVICE WORKING TEMPERATURE IN KELVIN
device_temperature = 0.01

# DEFINE PHYSICAL MATERIALS
semiconductor      = mt.Si     # Silicon as semiconductor
dielectric      = mt.SiO2   # Silicon dioxide as the dielectric
```

```python
metal_workfunction  = 4.33 * ct.e   # Titanium workfunction (Joules)

# DEFINE DOPING AND VOLTAGES
default_n_doping = 1e18*1e6
default_plunger_voltage = 0.5
n_doping    = float(os.getenv("N_DOPING",  default_n_doping))
V_plunger_1 = float(os.getenv("V_PLUNGER", default_plunger_voltage))
V_plunger_2 = V_plunger_1
V_barrier_1 = -0.1
V_barrier_2 = -0.1 #-0.8
V_barrier_3 = V_barrier_1

# DEFINE THE DIFFERENT DOPING CONCENTRATIONS TO BE USED. REFER TO SIMULATION RESULTS FROM SENTAURUS
doping_npp = n_doping           # Define the n_doping as the peak concentration, localized at the surface
doping_np  = n_doping * 1e-1    # Define the second doping concentration as one order of magnitude less wrt peak concentration
doping_n   = n_doping * 1e-2    # Define the third doping concentration as two order of magnitude less wrt peak concentration

# DEFINE BACKGATE PROPERTIES
use_backgate = False    # Backgate only defined if this is set True
V_backgate = -0.5
backgate_doping = "n"
backgate_dose = 1e15*1e6
backgate_binding_energy = 46e-3*ct.e    # Dopant ionization energy for phosphorus donors

# CHOOSE IF SAVING THE HDF5 FILE, USED FOR THE TRANSPORT LAYER SIMULATION
save_hdf5 = True

# CHOOSE IF SAVING THE EIGENSTATES IN .VTU FORMAT
save_psi0_vtu = True
save_psi1_vtu = False

# CHOOSE IF COMPUTE CHARGE STABILITY DIAGRAM
compute_csd = False

# START BY RESET THE DEVICE CONFIGURATION SCORE
score = 0


```

```


# ------------------------------------------------------------- #
# LOADING THE MESH                                              #
# ------------------------------------------------------------- #

# SET MESH SCALING FACTOR TO NANOMETERS
# If importing into GMSH a .step/.iges file from Fusion360 the
    workspace unit of the latter should be set to millimiters!
scaling = 1e-9

# DEFINE MESH VIA SCALING FACTOR AND MESH PATH
mesh = Mesh(scaling, path_mesh)

# GLOBAL NODES OF FULL DEVICE
x = mesh.glob_nodes[:, 0]
y = mesh.glob_nodes[:, 1]
z = mesh.glob_nodes[:, 2]

# FULL DIMENSIONS OF THE DEVICE IN NANOMETERS
dim_x = np.abs(np.min(x)*1e9)+np.abs(np.max(x)*1e9)
dim_y = np.abs(np.min(y)*1e9)+np.abs(np.max(y)*1e9)
dim_z = np.abs(np.min(z)*1e9)+np.abs(np.max(z)*1e9)

# DEFINE THE DEVICE LENGTH AS THE TOTAL LENGTH MINUS THE LATERAL
    OXIDE
device_length = dim_x*1e-9 - 2*lateral_oxide

# CREATE DEVICE FROM MESH AND SET CONFINED CARRIERS TO ELECTRONS
d = Device(mesh, conf_carriers = "e")
d.set_temperature(device_temperature)
d.statistics = "FD_approx"  # Aproximated Fermi-Dirac distribution




# ------------------------------------------------------------- #
# DEFINE DEVICE PHYSICAL PROPERTIES                             #
# ------------------------------------------------------------- #

# DEFINE VOLUME PHYSICAL CONDITIONS
d.new_region("semi_channel", semiconductor)

d.new_region("source_npp", semiconductor, pdoping=0, ndoping=
    doping_npp)
d.new_region("source_np",  semiconductor, pdoping=0, ndoping=
    doping_np)
d.new_region("source_n",   semiconductor, pdoping=0, ndoping=
    doping_n)
```

```

d.new_region("drain_npp",  semiconductor, pdoping=0, ndoping=
    doping_npp)
d.new_region("drain_np",   semiconductor, pdoping=0, ndoping=
    doping_np)
d.new_region("drain_n",    semiconductor, pdoping=0, ndoping=
    doping_n)

d.new_region("diel_oxide",     dielectric)
d.new_region("diel_barrier_1", dielectric)
d.new_region("diel_barrier_2", dielectric)
d.new_region("diel_barrier_3", dielectric)
d.new_region("diel_plunger_1", dielectric)
d.new_region("diel_plunger_2", dielectric)

# DEFINE SURFACES BOUNDARY CONDITIONS
d.new_ohmic_bnd("ohmicbnd_drain")
d.new_ohmic_bnd("ohmicbnd_source")

d.new_gate_bnd("gatebnd_barrier_1", V_barrier_1,
    metal_workfunction)
d.new_gate_bnd("gatebnd_barrier_2", V_barrier_2,
    metal_workfunction)
d.new_gate_bnd("gatebnd_barrier_3", V_barrier_3,
    metal_workfunction)
d.new_gate_bnd("gatebnd_plunger_1", V_plunger_1,
    metal_workfunction)
d.new_gate_bnd("gatebnd_plunger_2", V_plunger_2,
    metal_workfunction)

if use_backgate:
     d.new_frozen_bnd("gatebnd_backgate", V_backgate, semiconductor
         , backgate_dose, backgate_doping, backgate_binding_energy)

# DEFINE THE DOT REGION AS A LIST OF REGION LABELS THAT COMPOSE
    THE DOT MEDIUM AND BARRIERS
dot_region = ["semi_channel", "diel_oxide", "diel_barrier_1", "
    diel_barrier_2", "diel_barrier_3", "diel_plunger_1", "
    diel_plunger_2"]

# SET UP THE DOT REGION IN WHICH NO CLASSICAL CHARGE IS ALLOWED
d.set_dot_region(dot_region)




# -------------------------------------------------------------- #
# NON-LINEAR POISSON SOLVER                                      #
# -------------------------------------------------------------- #
```

```python

# CREATE A POISSON SOLVER PARAMETERS OBJECT
params_poisson = PoissonSolverParams()

# The tolerance attribute tol specifies the maximum acceptable
    potential difference (in volts)
# between two successive self-consistent-loop iterations
params_poisson.tol = 1e-5

# CREATE AND SOLVE A NON-LINEAR POISSON SOLVER
s = PoissonSolver(d, solver_params=params_poisson)
s.solve()

# SAVE POISSON RESULTS IN THE HDF5 FILE
if save_hdf5:
    io.save(str(path_hdf5), {"n": d.n/1e6, "p": d.p/1e6, "phi": d.
        phi, "EC": d.cond_band_edge()/ct.e, "EV": d.
        vlnce_band_edge()/ct.e})

# GET (OR COMPUTE ON THE DESIRED GATE IF NO DATA IS AVAILABLE IN
    THE lever_arm.mat FILE) THE DEVICE LEVERARM
leverarm = get_lever_arm(d, "gatebnd_plunger_1")
print(f"Lever arm = {leverarm*1000:.2f}meV")




# ---------------------------------------------------------------- #
# SCHRODINGER SOLVER                                               #
# ---------------------------------------------------------------- #

# GET THE POTENTIAL ENERGY FROM THE BAND EDGE FOR USAGE IN THE
    SCHRODINGER SOLVER
d.set_V_from_phi()

# CREATE A SUBMESH INCLUDING ONLY THE DOT REGION AND A SUBDEVICE
    FOR THE LATTER
submesh = SubMesh(d.mesh, dot_region)
subdevice = SubDevice(d, submesh)

# CREATE A SCHRODINGER SOLVER PARAMETERS OBJECT
params_schrod = SchrodingerSolverParams()
params_schrod.num_states = 4     # Specify the number of
    eigenstates and energies to consider in the diagonalization of
     the dot Hamiltonian
params_schrod.tol = 1e-12        # Set the tolerance for
    convergence on energies in electron-volts

# CREATE AND SOLVE A SCHRODINGER SOLVER
```

```
schrod_solver = SchrodingerSolver(subdevice)
schrod_solver.solve()

# PRINT EIGENENERGIES
subdevice.print_energies()



# ------------------------------------------------------------- #
# SAVE AND PLOT SCHRODINGER RESULTS                              #
# ------------------------------------------------------------- #

# GLOBAL NODES FOR THE SUBDEVICE
xdot = submesh.glob_nodes[:, 0]
ydot = submesh.glob_nodes[:, 1]
zdot = submesh.glob_nodes[:, 2]

# START BY FINDING MAX PROBABILITY FOR Z AXIS UNDER THE PLUNGER
    GATE NEAR THE SOURCE
# THEN PLACE X AND Y LINECUTS AT THAT Z COORDINATE
psi0 = np.abs(subdevice.eigenfunctions[:, 0])**2

# WAVEFUNCTION AND POTENTIAL ENERGY IN THE ZoY PLANE
beginz = (plunger_x_coordinate, 0, np.max(zdot))
endz =   (plunger_x_coordinate, 0, np.min(zdot))
psiposz, psiz0 = linecut(submesh, psi0, beginz, endz)
psiposz = -psiposz
psiposz += np.max(zdot)
V_plungerosz, Vz = linecut(mesh, d.V, beginz, endz)
V_plungerosz = -V_plungerosz
V_plungerosz += np.max(z)

# FIND MAXIMA OF PSIO_Z AND PSIO_X
dot_z_position = psiposz[np.argmax(psiz0)]

# WAVEFUNCTION AND POTENTIAL ENERGY IN THE XoZ PLANE
beginx = (-device_length/2, 0, dot_z_position)
endx =   (device_length/2, 0, dot_z_position)
psiposx, psix0 = linecut(submesh, psi0, beginx, endx)
psiposx += (np.min(xdot) + lateral_oxide)
V_plungerosx, Vx = linecut(mesh, d.V, beginx, endx)
V_plungerosx += (np.min(x) + lateral_oxide)

# WAVEFUNCTION AND POTENTIAL ENERGY IN THE YoX PLANE
beginy = (plunger_x_coordinate, np.min(y), dot_z_position)
endy =   (plunger_x_coordinate, np.max(y), dot_z_position)
psiposy, psiy0 = linecut(submesh, psi0, beginy, endy)
psiposy += np.min(ydot)
V_plungerosy, Vy = linecut(mesh, d.V, beginy, endy)
```

```
V_plungerosy += np.min(y)

# FIND MAXIMA OF PSIO_X
normalized_psix0 = psix0/np.max(psix0)

dot1_x, dot1_psix0 = crop_around_coordinate(psiposx,
    normalized_psix0, -np.abs(plunger_x_coordinate),
    plunger_length)
dot2_x, dot2_psix0 = crop_around_coordinate(psiposx,
    normalized_psix0, np.abs(plunger_x_coordinate), plunger_length
    )

dot1_x_position = dot1_x[np.argmax(dot1_psix0)]
dot2_x_position = dot2_x[np.argmax(dot2_psix0)]

localized_dot=False
if is_within_percentage(np.abs(dot1_x_position*1e9), np.abs(
    plunger_x_coordinate*1e9), 20):
     localized_dot=True

print(f"Quantum dots found at z = {dot_z_position*1e9:.2f} nm")

print(f"1st quantum dot found at x = {dot1_x_position*1e9:.2f} nm
    vs plunger at {-np.abs(plunger_x_coordinate*1e9):.2f} nm")
print(f"2nd quantum dot found at x = {dot2_x_position*1e9:.2f} nm
    vs plunger at {np.abs(plunger_x_coordinate*1e9):.2f} nm")


dot1_x_confinement = x_for_threshold(psiposx, normalized_psix0, -
    np.abs(plunger_x_coordinate), plunger_length)
dot2_x_confinement = x_for_threshold(psiposx, normalized_psix0, np
    .abs(plunger_x_coordinate), plunger_length)

print(f"1st quantum dot confinement = {dot1_x_confinement*100:.2f
    }% (DQD) - {dot1_x_confinement*200:.2f}% (SQD)")
print(f"2nd quantum dot confinement = {dot2_x_confinement*100:.2f
    }% (DQD) - {dot2_x_confinement*200:.2f}% (SQD)")
print(f"Delta confinement = {np.abs((dot1_x_confinement/0.5)-(
    dot2_x_confinement/0.5))*500:.2f}%")

if localized_dot:
     print("[OK] Dot is localized under the plunger gate!")
else:
     print("[KO] Dot IS NOT localized under the plunger gate!")

# COMPUTE Z-AXIS CUMULATIVE CONFINEMENT LEVEL
z_thresh_2N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
    threshold=0.99)*1e-9
```

```
z_thresh_3N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
    threshold=0.999)*1e-9
z_thresh_4N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
    threshold=0.9999)*1e-9
z_thresh_5N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
    threshold=0.99999)*1e-9
z_thresh_6N = z_for_threshold(psiposz*1e9, psiz0/np.max(psiz0),
    threshold=0.999999)*1e-9
print(f"2N threshold for quantum dot at z = {z_thresh_2N*1e9:.2f}
    nm")
print(f"3N threshold for quantum dot at z = {z_thresh_3N*1e9:.2f}
    nm")
print(f"4N threshold for quantum dot at z = {z_thresh_4N*1e9:.2f}
    nm")
print(f"5N threshold for quantum dot at z = {z_thresh_5N*1e9:.2f}
    nm")
print(f"6N threshold for quantum dot at z = {z_thresh_6N*1e9:.2f}
    nm")


# SAVE DEVICE RESULTS IN THE .MAT FILE
savemat(path_results_mat, {
"x":psiposx/1e-9,
"y":psiposy/1e-9,
"z":psiposz/1e-9,
"psi0_x":psix0,
"psi0_y":psiy0,
"psi0_z":psiz0,
"V_x":Vx/ct.e,
"V_y":Vy/ct.e,
"V_z":Vz/ct.e,
"doping":n_doping,
"V_plunger":V_plunger_1,
"V_barrier":V_barrier_1,
"leverarm":leverarm,
"dot_z":dot_z_position/1e9,
"dot_z_threshold_2N":z_thresh_2N/1e9,
"dot_z_threshold_3N":z_thresh_3N/1e9,
"dot_z_threshold_4N":z_thresh_4N/1e9,
"dot_z_threshold_5N":z_thresh_5N/1e9,
"dot_z_threshold_6N":z_thresh_6N/1e9,
"energies":subdevice.energies/ct.e
})

# SAVE SCHRODINGER RESULTS IN .VTU FORMAT
if save_psi0_vtu:
    io.save(path_psi0, np.abs(subdevice.eigenfunctions[:, 0])**2,
        submesh)
if save_psi1_vtu:
    io.save(path_psi1, np.abs(subdevice.eigenfunctions[:, 1])**2,
        submesh)

# UPDATE DEVICE CONFIGURATION SCORE
score += (dot1_x_confinement/0.5)*100
score += (dot2_x_confinement/0.5)*100
score += 50*np.exp(-5*np.abs(np.abs(dot1_x_position*1e9)-np.abs(
    plunger_x_coordinate*1e9)))
score += 50*np.exp(-5*np.abs(np.abs(dot2_x_position*1e9)-np.abs(
    plunger_x_coordinate*1e9)))
score += 100*np.exp(-np.abs(dot_z_position*1e9))
score -= 100*np.exp(-5*np.abs((dot1_x_confinement/0.5)-(
    dot2_x_confinement/0.5)))

# ------------------------------------------------------------- #
# PLOT RESULTS                                                  #
# ------------------------------------------------------------- #

# DPI AND ORIGINAL DIMENSIONS FOR A SINGLE PLOT
dpi = 300
single_w_px = 1500
single_h_px = 1000

# 2 COLUMNS x 2 ROWS
fig_w_px = single_w_px * 2
fig_h_px = single_h_px * 2

fig = plt.figure(figsize=(fig_w_px/dpi, fig_h_px/dpi), dpi=dpi)

# SPACING
w_frac = 0.36
h_frac = 0.375

title = (
fr'{mesh_name}' + '\n' +
fr'{mesh.node_number} nodes, {dim_x:.1f}x{dim_y:.1f}x{dim_z:.1f}
    nm'
)
textstr = (
fr'T = {device_temperature:.2f} K' + '\n' +
fr'$dose_{{\mathrm{{n}}}}$ = ' + get_apex(n_doping) + ' $m^{{\
    mathrm{{-3}}}}$' + '\n' +
fr'$V_{{\mathrm{{plungers}}}}$ = {V_plunger_1:.2f} V' + '\n' +
fr'$V_{{\mathrm{{b\_lateral}}}}$ = {V_barrier_1:.2f} V' + '\n' +
fr'$V_{{\mathrm{{b\_central}}}}$ = {V_barrier_2:.2f} V'
)

text_desc=(
fr'$z_{{\mathrm{{dot}}}}$ = {dot_z_position*1e9:.2f} nm' + '\n' +
fr'2$\sigma$ = {z_thresh_2N*1e9:.2f} nm' + '\n' +
fr'$\alpha$ = {leverarm*1000:.1f} meV/V' + '\n' +
fr'score = {int(score)} pt'
)

text_conf=(
fr'$z_{{\mathrm{{dot}}}}$ = {dot_z_position*1e9:.2f} nm' + '\n' +
fr'2$\sigma$ = {z_thresh_2N*1e9:.2f} nm' + '\n' +
fr'3$\sigma$ = {z_thresh_3N*1e9:.2f} nm' + '\n' +
fr'4$\sigma$ = {z_thresh_4N*1e9:.2f} nm' + '\n' +
fr'5$\sigma$ = {z_thresh_5N*1e9:.2f} nm' + '\n' +
fr'6$\sigma$ = {z_thresh_6N*1e9:.2f} nm'
)

# QUADRANT 1: DEVICE INFORMATIONS AND SIMULATION VARTIABLES
text_loc="X"
if localized_dot:
    text_loc=f"{int(score)}"

ax_text = fig.add_axes([0.02, 0.52, w_frac, h_frac])
ax_text.axis('off')
ax_text.text(0.5, 0.95, title, fontsize=15, ha='center', va='
    center', wrap=True)
ax_text.text(-0.025, 0.15, textstr, fontsize=14, ha='left', va='
    center', wrap=True)
ax_text.text(0.625, 0.15, text_desc, fontsize=14, ha='left', va='
    center', wrap=True)
img = mpimg.imread(path_image)
imagebox = OffsetImage(img, zoom=0.1)
ab = AnnotationBbox(imagebox, (0.5, 0.65), frameon=False, xycoords
    ='axes fraction')
ax_text.add_artist(ab)

# QUADRANT 2: LINECUT ALONG CHANNEL
y_percentage_label_coordinate = 0.5
ax1 = fig.add_axes([0.52, 0.52, w_frac, h_frac])
ax2 = ax1.twinx()
ax1.set_title(fr'Linecut along channel ($x$) @ $z_{{\mathrm{{dot
    }}}} = {dot_z_position*1e9:.2f}\,\mathrm{{nm}}$')
ax1.plot(psiposx / 1e-9, psix0/np.max(psix0), linewidth=2)
ax2.plot(V_plungerosx / 1e-9, Vx / ct.e, '--r')
ax1.grid()
ax1.set_xlabel("x [nm]")
ax1.set_ylabel(r"Normalized $|\Psi(x, 0, z_{dot})|^2 [m^{-3}]$",
    color='blue')
ax2.set_ylabel(r"$V [eV]$", color='red')
for label in ax1.get_yticklabels():
    label.set_color('blue')
for label in ax2.get_yticklabels():
```

```python
    label.set_color('red')
# Add confinement percentage and peak x location on each dot
ax2.text(((-np.abs(dot1_x_position)+device_length/2)/device_length), y_percentage_label_coordinate, f"{dot1_x_confinement*100:.1f}%",
         transform=ax1.transAxes,
         fontsize=10,
     zorder=10,
         verticalalignment='center',
         horizontalalignment='center',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white', alpha=0.8, edgecolor='blue'))
ax2.text(((np.abs(dot2_x_position)+device_length/2)/device_length), y_percentage_label_coordinate, f"{dot2_x_confinement*100:.1f}%",
         transform=ax1.transAxes,
         fontsize=10,
     zorder=10,
         verticalalignment='center',
         horizontalalignment='center',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white', alpha=0.8, edgecolor='blue'))
ax2.text(((dot1_x_position+device_length/2)/device_length), np.max(dot1_psix0)-0.05, f"{dot1_x_position*1e9:.1f}nm",
         transform=ax1.transAxes,
         fontsize=10,
     zorder=10,
         verticalalignment='center',
         horizontalalignment='center',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white', alpha=0.8, edgecolor='green'))
ax2.text(((dot2_x_position+device_length/2)/device_length), np.max(dot2_psix0)-0.05, f"{dot2_x_position*1e9:.1f}nm",
         transform=ax1.transAxes,
         fontsize=10,
     zorder=10,
         verticalalignment='center',
         horizontalalignment='center',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white', alpha=0.8, edgecolor='green'))

# QUADRANT 3: LINECUT ALONG Y
ax3 = fig.add_axes([0.02, 0.02, w_frac, h_frac])
ax4 = ax3.twinx()
ax3.set_title(fr'Linecut along $y$ @ $z_{{\mathrm{{dot}}}}$, $x_{{\mathrm{{plunger}}}}$')
ax3.plot(psiposy / 1e-9, psiy0/np.max(psiy0), linewidth=2)
ax4.plot(V_plungerosy / 1e-9, Vy / ct.e, '--r')
ax3.grid()
```

```python
ax3.set_xlabel("y [nm]")
ax3.set_ylabel(r"Normalized $|\Psi(x_{plunger} , y, z_{dot})|^2 [m
    ^{-3}]$", color='blue')
ax4.set_ylabel(r"$V [eV]$", color='red')
for label in ax3.get_yticklabels():
    label.set_color('blue')
for label in ax4.get_yticklabels():
    label.set_color('red')

# QUADRANT 4: LINECUT ALONG Z
ax5 = fig.add_axes([0.52, 0.02, w_frac, h_frac])
ax6 = ax5.twinx()
ax5.set_title(fr'Linecut along $z$ @ $x_{{\mathrm{{plunger}}}} = {
    plunger_x_coordinate*1e9:.2f}\,\mathrm{{nm}}$')
ax5.plot(psiposz / 1e-9, psiz0/np.max(psiz0), linewidth=2)
ax6.plot(V_plungerosz / 1e-9, Vz / ct.e, '--r')
ax5.grid()
ax5.set_xlabel("z [nm]")
ax5.set_ylabel(r"Normalized $|\Psi(x_{plunger} , 0, z)|^2 [m^{-3}]
    $", color='blue')
ax6.set_ylabel(r"$V [eV]$", color='red')
for label in ax5.get_yticklabels():
    label.set_color('blue')
for label in ax6.get_yticklabels():
    label.set_color('red')
ax6.text(0.02, 0.98, text_conf,
         transform=ax6.transAxes,
         fontsize=11,
     zorder=10,
         verticalalignment='top',
         horizontalalignment='left',
         bbox=dict(boxstyle='round,pad=0.2', facecolor='white',
            alpha=0.7, edgecolor='none'))

# SAVE IMAGE
if batch == "True":
    path_results_img = "../" + text_loc + "_" + f"V{V_plunger_1}_N
        {int(n_doping):.0e}" + ".png"
plt.savefig(path_results_img, dpi=dpi, bbox_inches='tight')

# IF SELECTED, COMPUTE CHARGE STABILITY DIAGRAM WITH THE CURRENT
    DEVICE CONFIGURATION (SLOW)
if compute_csd:
    compute_charge_stability_diagram(subdevice, results_dir)
```

Listing B.1: `sim_dqd.py` - Main simulation script for the `device` layer in a DQD device. Its operation is explained in detail in Chapter 6.

B.2 charge_stability_diagram.py

```
import os
import sys
sys.path.append('/opt/qtcad-1.4.3/qtcad/')

from qtcad.device import constants as ct
from qtcad.device.mesh3d import Mesh, SubMesh
from qtcad.device.analysis import linecut
from qtcad.device import io
from qtcad.device import analysis
from qtcad.device import materials as mt
from qtcad.device import Device, SubDevice
from qtcad.device.poisson import Solver as PoissonSolver
from qtcad.device.poisson import SolverParams as
    PoissonSolverParams
from qtcad.device.schrodinger import Solver as SchrodingerSolver
from qtcad.device.schrodinger import SolverParams as
    SchrodingerSolverParams
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
import pathlib
import numpy as np
import math
import pickle
from qtcad.device.many_body import SolverParams
from qtcad.transport.mastereq import seq_tunnel_curr
from qtcad.transport.junction import Junction
from progress.bar import ChargingBar as Bar

sys.path.append(os.path.join(os.path.dirname(__file__), "tools"))
from get_leverarm import get_lever_arm

def compute_charge_stability_diagram(device, results_dir):
    print("COMPUTING CHARGE STABILITY DIAGRAM - COULOMB DIAMONDS")
    many_body_solver_params = SolverParams()
    many_body_solver_params.num_states = 2  # Number of levels to
        keep
    many_body_solver_params.n_degen = 2     # Spin degenerate
        system
    many_body_solver_params.alpha = get_lever_arm(device, "
        gatebnd_plunger_1")  # Lever arm
    jc = Junction(device, many_body_solver_params=
        many_body_solver_params)

    print('chemical potentials (eV):', jc.chem_potentials/ct.e)
    print('positions of Coulomb peaks relative to reference value
       (V):',
        jc.coulomb_peak_pos)

    # Set a near-zero source-drain bias
    jc.setVs(0.0001)
    jc.setVd(-0.0001)

    jc.set_source_broad_func(10e6)
    jc.set_drain_broad_func(10e6)

    v_gate_rng = np.linspace(0, 1, num=500)

    # Calculate the current for each plunger gate potential
    vec_Il = []
    for i in range(v_gate_rng.size):        # loop over gate
       voltage
        v_gate = v_gate_rng[i]
        jc.setVg(v_gate)
        Il,Ir,prob = seq_tunnel_curr(jc)    # transport
           calculation
        vec_Il += [Il]

    fig = plt.figure(figsize=(8,5))
    ax = fig.add_subplot(1,1,1)
    ax.set_xlabel("Gate potential (V)")
    ax.set_ylabel("Current (pA)")
    ax.plot(v_gate_rng, np.array(vec_Il)/1e-12)
    path_fig = str(results_dir / "coulomb_peaks.png")
    plt.savefig(path_fig)
    #plt.show()

    ### Charge-stability diagram ###
    v_gate_rng = np.linspace(-1, 1, num=100)
    v_drain_rng = np.linspace(-0.05, 0.05, num=100)

    diam = np.zeros((v_gate_rng.size,v_drain_rng.size), dtype=
       float)

    number_grid_points = v_gate_rng.size * v_drain_rng.size #
       number of sampled grid points
    progress_bar = Bar("Computing charge-stability diagram", max =
        number_grid_points) # initialize progress bar

    for i in range(v_gate_rng.size):        # loop over gate
       voltage
        v_gate = v_gate_rng[i]
        jc.setVg(v_gate)

        for j in range(v_drain_rng.size):      # loop over drain voltage
            v_drain = v_drain_rng[j]
            jc.setVd(v_drain)
            Il,Ir,prob = seq_tunnel_curr(jc)    # transport calculation
            diam[i,j] = Il
            progress_bar.next()

    progress_bar.finish()

    # Postprocess the current to turn it into an image to be plotted
    current_image = np.flip(np.transpose(np.abs(diam)), axis=0)

    Vtop_2 = 1

    # plot results
    fig, axs = plt.subplots()
    axs.set_ylabel('$V_d(V)$',fontsize=16)
    axs.set_xlabel('$V_g(V)$',fontsize=16)
    diff_cond_map = axs.imshow(current_image/1e-12, interpolation='bilinear',
            extent=[Vtop_2+v_gate_rng[0], Vtop_2+v_gate_rng[-1],
                    v_drain_rng[0], v_drain_rng[-1]], aspect="auto", cmap="Purples")
    fig.colorbar(diff_cond_map, ax=axs, label="Current (pA)")
    fig.tight_layout()
    path_fig = str(results_dir / "charge_stability_diagram.png")
    plt.savefig(path_fig)
```

Listing B.2: `charge_stability_diagram.py` - This script can be called from the function `compute_charge_stability_diagram` at the end of the main simulation script for the device package. It takes the device and computes the charge stability diagram.

B.3 particle_addition_spectrum.py

```python
import os
import sys
sys.path.append('/opt/qtcad-1.4.3/qtcad/')

from qtcad.device import constants as ct
from qtcad.device.mesh3d import Mesh, SubMesh
from qtcad.device.analysis import linecut
from qtcad.device import io
from qtcad.device import analysis
from qtcad.device import materials as mt
from qtcad.device import Device, SubDevice
from qtcad.device.poisson import SolverParams as
    PoissonSolverParams
from qtcad.device.schrodinger import SolverParams as
    SchrodingerSolverParams
from qtcad.device.leverarm import Solver as LeverArmSolver
from qtcad.device.leverarm import SolverParams as
    LeverArmSolverParams
from qtcad.device.many_body import Solver as ManyBodySolver
from qtcad.device.many_body import SolverParams as
    ManyBodySolverParams
from qtcad.device.leverarm_matrix import Solver as LeverArmSolver
from qtcad.device.leverarm_matrix import SolverParams as
    LeverArmSolverParams
from qtcad.transport.junction import Junction
from qtcad.transport.mastereq import add_spectrum
from progress.bar import ChargingBar as Bar
from scipy.io import savemat, loadmat
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
import matplotlib.image as mpimg
import pathlib
from pathlib import Path
import numpy as np
import math


def get_add_spectrum(junc, gate_labels, gate_biases, temperature,
    verbose=True):

    # Set list of biases with drain potential + increment
    junc.set_biases(gate_labels,gate_biases, verbose=verbose)

    # Calculate response function
    out = add_spectrum(junc, temperature=temperature)
    return out


def compute_particle_addition_spectrum(device, dot_region, V_plunger_1, V_plunger_2, results_dir):
    print("COMPUTING CHARGE STABILITY DIAGRAM - PARTICLE ADDITION SPECTRUM")
    # Instantiate the voltage bias vector
    bias_vector = np.array([V_plunger_1, V_plunger_2])

    # Bias labels
    gate_labels = ["gatebnd_plunger_1", "gatebnd_plunger_2"]

    params_poisson = PoissonSolverParams()
    params_poisson.tol = 1e-5
    params_schrod = SchrodingerSolverParams()
    params_schrod.num_states = 4    # Specify the number of eigenstates and energies to consider in the diagonalization of the dot Hamiltonian
    params_schrod.tol = 1e-12       # Set the tolerance for convergence on energies in electron-volts

    # Solver params for the LeverArmSolver
    lam_params = LeverArmSolverParams()
    lam_params.pot_solver_params = params_poisson
    lam_params.schrod_solver_params = params_schrod

    # Instantiate lever arm matrix solver
    slv = LeverArmSolver(device, gate_labels, bias_vector, dot_region=dot_region,
    solver_params=lam_params)

    # Calculate the lever arm matrix
    bias_increment = 1e-3
    lever_arm_matrix = slv.solve(bias_increment=bias_increment)

    # Print and save the lever arm matrix
    print("Lever arm matrix")
    print(lever_arm_matrix)
    np.save(results_dir/"lever_arm_matrix.npy", lever_arm_matrix)

    submesh = SubMesh(device.mesh, dot_region)
    subdevice = SubDevice(device, submesh)

    # Calculate the Coulomb interaction matrix
    # Instantiate many-body solver
    num_states = 4
    many_body_solver_params = ManyBodySolverParams()
    many_body_solver_params.n_degen = 2
    many_body_solver_params.num_states = num_states
    slv = ManyBodySolver(subdevice, solver_params=
       many_body_solver_params)

    # Compute, save, and print Coulomb interaction matrix without
       overlap terms
    coulomb_no_overlap = slv.get_coulomb_matrix(overlap=False,
       verbose=True)
    np.save(results_dir/"coulomb_mat_no_overlap.npy",
       coulomb_no_overlap)
    print("Coulomb interaction matrix (eV)")
    print(coulomb_no_overlap/ct.e)

    # Set the junction temperature to 10 K (instead of 100 mK) to
       make lines
    # in the charge stability diagram thicker and decrease the
       resolution
    # required to observe them
    temperature_spec = 10

    # Contact labels
    gate_labels = ["ohmicbnd_source", "gatebnd_plunger_1", "
       gatebnd_plunger_2",
    "ohmicbnd_drain"]

    # Instantiate a junction
    many_body_solver_params.energies = subdevice.energies
    many_body_solver_params.overlap = False
    many_body_solver_params.coulomb_mat = coulomb_no_overlap
    many_body_solver_params.alpha = lever_arm_matrix
    jc = Junction(many_body_solver_params =
       many_body_solver_params,
    temperature=temperature_spec, contact_labels=gate_labels)

    # Base source and drain potentials
    source_potential = 0
    drain_potential = 0

    # Extremal values of the gate biases
    min_gate_1_bias = -0.5 ; max_gate_1_bias = 0.5
    min_gate_2_bias = -0.5 ; max_gate_2_bias = 0.5

    gate_1_biases = np.linspace( min_gate_1_bias, max_gate_1_bias,
        90)
    gate_2_biases = np.linspace( min_gate_2_bias, max_gate_2_bias,
        90)
    add_spectrum_mat = np.zeros((len(gate_1_biases),len(
       gate_2_biases)))
```

```python

    bartitle = "Calculating charge stability diagram."
    progress_bar = Bar(bartitle, max = len(gate_1_biases)*len(gate_2_biases))
    for idx_1, gate_1_bias in enumerate(gate_1_biases):
        for idx_2, gate_2_bias in enumerate(gate_2_biases):
            add_spectrum_mat[idx_1,idx_2] =\
                    get_add_spectrum(jc, gate_labels, np.array([source_potential,
                    gate_1_bias, gate_2_bias, drain_potential]), temperature_spec,
                    verbose=False)
            np.savetxt(results_dir / "addition_spectrum.txt", add_spectrum_mat)
            progress_bar.next()

    progress_bar.finish()

    # Plot the particle addition spectrum
    # Transpose and flip the matrix to turn into an image with bias_1 along x
    # and bias_2 along y
    add_spec_to_plot = np.flip(np.transpose(add_spectrum_mat),axis=0)
    fig, axs = plt.subplots()
    axs.set_xlabel('$V_{g1}$ (V)',fontsize=16)
    axs.set_ylabel('$V_{g2}$ (V)',fontsize=16)
    vg1_ref = V_plunger_1; vg2_ref = V_plunger_2;
    diff_conds_map = axs.imshow(add_spec_to_plot/np.max(add_spec_to_plot),
        cmap="jet", interpolation='bilinear',
        extent=[min_gate_1_bias+vg1_ref, max_gate_1_bias+vg1_ref,
                min_gate_2_bias+vg2_ref, max_gate_2_bias+vg2_ref], aspect="auto")
    fig.colorbar(diff_conds_map, ax=axs, label="Response (arb. units)")
    fig.tight_layout()
    plt.savefig("particle_addition_spectrum.png", dpi=300, bbox_inches='tight')
```

Listing B.3: `particle_addition_spectrum.py` - This script can be called from the function `compute_particle_addition_spectrum` at the end of the main simulation script for the device package. It takes the device of a double quantum dot and computes the particle addition spectrum.

Appendix C

Python Additional Scripts

C.1 Batch Runner

```python
import json
import os
import itertools
import subprocess
from concurrent.futures import ProcessPoolExecutor
from pathlib import Path
import sys

sys.path.append(os.path.join(os.path.dirname(__file__), "tools"))
from mat_merge import merge_dot_results
from get_leverarm import get_lever_arm


WORKERS = int(os.cpu_count()/2)

# DEFINE ENVIRONMENTAL VARIABLES
env = os.environ.copy()

# DEFINE PARAMETERS TO WEEP
plunger_values = [0.5, 0.6, 0.8, 1.0]
doping_values = [1e15*1e6,1e16*1e6,1e17*1e6,1e18*1e6,1e19*1e6,1e20
    *1e6]

# COMBINATIONS
combinations = list(itertools.product(plunger_values,
    doping_values))

# BASE OUTPUT FOLDER
base_output = Path("batch_results")
base_output.mkdir(exist_ok=True)

```

```python
# BE SURE TO HAVE LEVER ARM COMPUTED BEFORE PARALLELIZATION, OTHERWISE COMMENT IN SIMULATION SCRIPT

def run_simulation(plunger, doping):
    # Define directory name for the single simulation
    label = f"V{plunger}_N{int(doping):.0e}"
    output_dir = base_output / label
    output_dir.mkdir(exist_ok=True)

    env["V_PLUNGER"] = str(plunger)
    env["N_DOPING"] = str(doping)

    env["BATCH"] = str("True")

    print(f"[INFO] Running sim: {label}")

    subprocess.run(
        ["python", str(Path(__file__).parent/"sim_v4.py")],
        cwd=output_dir,  # save files inside simulation directory
        env=env,
    stdout=subprocess.DEVNULL
    )

# PARALLELIZZAZION
if __name__ == "__main__":
    with ProcessPoolExecutor(max_workers=WORKERS) as executor:
        futures = [executor.submit(run_simulation, v, n) for v, n in combinations]
        for f in futures:
            f.result()

merge_dot_results()
```

Listing C.1: `batch_runner.py` - Pilot script for the simulation one. It enables process-level parallelization to carry out multiple simulations, each corresponding to a configuration defined by a set of different values for various variables. Its operation is explained in detail in Section 6.2.

C.2 Process Variation Tool

```python
import gmsh
import os
import sys


def get_surfaces_of_volume_by_label(label):
    # Find the volume with the given label (e.g. "oxide")
    entities = gmsh.model.getEntities(dim=3)
    for dim, tag in entities:
        name = gmsh.model.getEntityName(dim, tag)
        if name == label:
            oxide_tag = (dim, tag)
            break
    else:
        raise ValueError(f"[ERROR] Volume with label '{label}' not
            found.")

    # Get the surfaces that form the boundary of the volume
    surfaces = gmsh.model.getBoundary([oxide_tag], oriented=False,
        recursive=False)

    # Print the surface tags
    print(f"\n[SEARCH] Surfaces forming the volume '{label}':")
    for dim, tag in surfaces:
        if dim == 2:
            name = gmsh.model.getEntityName(dim, tag)
            print(f"- Surface tag = {tag}{f' -> {name}' if name
                else ''}")

    return [tag for dim, tag in surfaces if dim == 2]


# === Get all Shapes names from IGES ===
def get_shapes_dict():
    entities = gmsh.model.getEntities()
    shapes_dict = {}
    for dim, tag in entities:
        try:
            name = gmsh.model.getEntityName(dim, tag)
        except Exception:
            continue
        if name.startswith("Shapes/"):
            shape_name = name.split("/")[-1]
            if shape_name not in shapes_dict:
                shapes_dict[shape_name] = []
            shapes_dict[shape_name].append((dim, tag))
    return shapes_dict

# === Get the tag (3D) from a name ===
def get_tag_by_label(label, shapes_dict):
    entries = shapes_dict.get(label, [])
    for dim, tag in entries:
        if dim == 3:
            return (3, tag)
    raise ValueError(f"No volume (dim=3) found with label '{label
       }'")

# === Boolean union of multiple shapes given the labels ===
def union_shapes_by_labels(labels, shapes_dict):
    if not labels:
        return []
    shapes = [get_tag_by_label(label, shapes_dict) for label in
       labels]
    current_union = [shapes[0]]
    for shape in shapes[1:]:
        current_union, _ = gmsh.model.occ.fuse(current_union, [
           shape])
    gmsh.model.occ.synchronize()
    return current_union


def get_z_max_of_label(label, shapes_dict):
    entry = shapes_dict.get(label, [])
    for dim, tag in entry:
        if dim == 3:
            bbox = gmsh.model.getBoundingBox(dim, tag)
            return bbox[5]  # z_max
    raise ValueError(f"No volume (dim=3) found with label '{label
       }'")

def add_surface(x, y, x_size, y_size, z, name=None):
    # Create a rectangular surface at position (x, y, z) with
       given size
    x0 = x - x_size / 2
    y0 = y - y_size / 2
    x1 = x + x_size / 2
    y1 = y + y_size / 2

    p1 = gmsh.model.occ.addPoint(x0, y0, z)
    p2 = gmsh.model.occ.addPoint(x1, y0, z)
    p3 = gmsh.model.occ.addPoint(x1, y1, z)
    p4 = gmsh.model.occ.addPoint(x0, y1, z)

    l1 = gmsh.model.occ.addLine(p1, p2)
    l2 = gmsh.model.occ.addLine(p2, p3)
    l3 = gmsh.model.occ.addLine(p3, p4)
    l4 = gmsh.model.occ.addLine(p4, p1)

    loop = gmsh.model.occ.addCurveLoop([l1, l2, l3, l4])
    surface_tag = gmsh.model.occ.addPlaneSurface([loop])

    gmsh.model.occ.synchronize()

    if name:
        gmsh.model.setEntityName(2, surface_tag, name)

    return surface_tag

def find_entity_by_label(label, dim=3):
    # Find an entity of a given dimension with the given label
    entities = gmsh.model.getEntities(dim)
    for d, tag in entities:
        name = gmsh.model.getEntityName(d, tag)
        if name == label:
            return (d, tag)
    raise ValueError(f"No entity found with label '{label}' and
       dimension {dim}")


def print_surface_tags_with_info(label):
    oxide_entity = find_entity_by_label(label, dim=3)

    # Find all the surfaces of the volume
    surfaces = gmsh.model.getBoundary([oxide_entity], oriented=
       False)

    print(f"\n[SEARCH] Surfaces of volume '{label}' (tag={
       oxide_entity[1]}):")
    for dim, tag in surfaces:
        if dim != 2:
            continue
        name = gmsh.model.getEntityName(dim, tag)
        bbox = gmsh.model.getBoundingBox(dim, tag)
        center = [(bbox[0] + bbox[3])/2, (bbox[1] + bbox[4])/2, (
           bbox[2] + bbox[5])/2]
        print(f"- Surface tag: {tag:3} {f'-> {name}' if name else
           ''} | center = {center}")


# === MAIN SCRIPT ===
def main(file_base):
    iges_file = f"{file_base}.iges"
    step_file = f"{file_base}_mod.step"
    output_file = "output.step"

    if not os.path.exists(iges_file):
        print(f"Error: IGES file '{iges_file}' not found.")
        sys.exit(1)

    gmsh.initialize()
    gmsh.option.setNumber("General.Terminal", 1)
    gmsh.model.add("modello_modificato")

    gmsh.model.occ.importShapes(iges_file)
    gmsh.model.occ.synchronize()

    shapes_dict = get_shapes_dict()
    print("[FOUND] Shapes found (name -> (dim, tag)):")
    for name, ents in shapes_dict.items():
        print(f" - {name}: {ents}")

    z = round(get_z_max_of_label("oxide", shapes_dict), 1)

    # === Merge some shapes ===
    labels_to_merge = ["plunger_1", "plunger_2", "barrier_1", "
       barrier_2", "barrier_3", "oxide"]
    merged = union_shapes_by_labels(labels_to_merge, shapes_dict)
    resulting_volumes = gmsh.model.occ.getEntities(dim=3)
    oxide_tag = 0
    for dim, tag in resulting_volumes:
        name = gmsh.model.getEntityName(dim, tag)
        if name == "Shapes/oxide":
            oxide_tag = tag

    # === Add a surface on top ===
    surf_tag = add_surface(0, 0, 20, 10, z, name="top_contact")
    print(f"[DONE] Surface 'top_contact' added with tag {surf_tag}
       ")

    oxide_surface_tags = get_surfaces_of_volume_by_label("Shapes/
       oxide")
    print_surface_tags_with_info("Shapes/oxide")

    top_tag = 94

    fused_surfaces, _ = gmsh.model.occ.fragment([(2, surf_tag)],
       [(2, top_tag)])
    gmsh.model.occ.synchronize()

    gmsh.model.occ.remove([(2, surf_tag)])
    gmsh.model.occ.synchronize()

    gmsh.model.occ.remove([(2, top_tag)])
```

```python
    gmsh.model.occ.synchronize()

    # === Export STEP file ===
    gmsh.write(output_file)
    print(f"[EXPORT] Geometry exported to '{output_file}'")

    gmsh.finalize()

# === Run script from terminal ===
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python script.py base_file_name (without
            extension)")
        sys.exit(1)
    base_name = sys.argv[1]
    main(base_name)
```

Listing C.2: `remesh.py` - This script can serve as a basis for process variation simulations. However, it requires corrections since the addition of the gate surface does not properly merge with the oxide volume and the top surface. A detailed description of its working principle is provided in Section 10.1.

Bibliography

[1] David Aasen, Morteza Aghaee, Zulfi Alam, Mariusz Andrzejczuk, and Andrey Antipov et al. Roadmap to fault tolerant quantum computation using topological qubit arrays, 2025.

[2] Asp Isotopes. Technology | asp isotopes, 2024. Accessed: 2025-05-20.

[3] Max Born, Emil Wolf, A. B. Bhatia, P. C. Clemmow, D. Gabor, A. R. Stokes, A. M. Taylor, P. A. Wayman, and W. L. Wilcock. *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light.* Cambridge University Press, 7 edition, 1999.

[4] Encyclopaedia Britannica. Lepton, February 2024. Accessed: May 14, 2025.

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

[6] Anasua Chatterjee, Paul Stevenson, Silvano De Franceschi, Andrea Morello, Nathalie P. de Leon, and Ferdinand Kuemmeth. Semiconductor qubits in practice. *Nature Reviews Physics*, 3(3), 2021.

[7] Claude Cohen-Tannoudji, Bernard Diu, and Franck Laloë. *Quantum Mechanics.* John Wiley & Sons, 2005.

[8] Wikipedia contributors. Born rule, 2025. Accessed: 2025-08-28.

[9] Wikipedia contributors. Debye length, 2025. Accessed: 2025-08-28.

[10] Wikipedia contributors. Plasma electrolytic oxidation, 2025. Accessed: 2025-08-28.

[11] Wikipedia contributors. Qubit, 2025. Accessed: 2025-08-28.

[12] Wikipedia contributors. Schrödinger equation, 2025. Accessed: 2025-08-28.

[13] Davide Costa. Definition of compact models for the simulation of spin qubits in semiconductor quantum dots, 2022.

[14] David P. DiVincenzo. Quantum computation. *Science*, 270(5234):255–261, 1995.

[15] David P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 2000.

[16] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47:777–780, May 1935.

[17] J. M. Elzerman, R. Hanson, L. H. Willems van Beveren, B. Witkamp, L. M. K. Vandersypen, and L. P. Kouwenhoven. Single-shot read-out of an individual electron spin in a quantum dot. *Nature*, 430(6998):431–435, 2004.

[18] Yaakov Y. Fein, Philipp Geyer, Patrick Zwick, Filip Kiałka, Sebastian Pedalino, Marcel Mayor, Stefan Gerlich, and Markus Arndt. Quantum superposition of molecules beyond 25 kda. *Nature Physics*, 15(12):1242–1245, 2019.

[19] David J. Griffiths. *Introduction to Quantum Mechanics*. Pearson Prentice Hall, Upper Saddle River, NJ, 2nd edition, 2004.

[20] Wonill Ha, Sieu D. Ha, Maxwell D. Choi, Yan Tang, Adele E. Schmitz, Mark P. Levendorf, Kangmu Lee, James M. Chappell, Tower S. Adams, Daniel R. Hulbert, Edwin Acuna, Ramsey S. Noah, Justine W. Matten, Michael P. Jura, Jeffrey A. Wright, Matthew T. Rakher, and Matthew G. Borselli. A flexible design platform for si/sige exchange-only qubits with low disorder. *Nano Letters*, 22(3):1443–1448, November 2021.

[21] Jianhua He. *Electron Spin Resonance on Si/SiGe Quantum Dots*. Ph.d. dissertation, Princeton University, Princeton, NJ, 2012.

[22] Y. Homma, A. Yajima, and S. Harada. A new soi (silicon-on-insulator) mos structure for vlsi's. *IEEE Journal of Solid-State Circuits*, 17(1):142–146, 1982.

[23] Nanoacademic Technologies Inc. Qtcad®: a computer-aided design tool for quantum-technology hardware. Web page, 2025. Accessed: 2025-09-08.

[24] Gregg Jaeger. What in the (quantum) world is macroscopic? *American Journal of Physics*, 82(9):896–905, 09 2014.

[25] B. E. Kane. A silicon-based nuclear spin quantum computer. *Nature*, 393(6681):133–137, 1998.

[26] M. A. Kastner. The single-electron transistor. *Rev. Mod. Phys.*, 64:849–858, Jul 1992.

[27] Jing Li, Jiayi Guo, Zhongchao Zhou, Rui Xu, Lina Xu, Yihong Ding, Hongping Xiao, Xinhua Li, Aidong Li, and Guoyong Fang. Atomic layer deposition mechanism of hafnium dioxide using hafnium precursor with amino ligands and water. *Surfaces and Interfaces*, 2023. Published online December 13, 2023.

[28] Olimpia Lombardi, Federico Holik, and Leonardo Vanni. What is shannon information? *Synthese*, 193(7):1983–2012, 2016.

[29] Nadya Mason. Electron transport in quantum dots, 2015. Accessed: 2025-08-28.

[30] Andrea Morello. Single spins in silicon carbide. *Nature Materials*, 14(2), 2015.

[31] Nanoacademic Technologies Inc. Double quantum dot stability diagram. `https://docs.nanoacademic.com/qtcad/tutorials/transport/double_dot_stability/`. Accessed: 2025-05-23.

[32] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge, 10 edition, 2010.

[33] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge, 10th anniversary edition edition, 2010.

[34] Thierry Ouisse. *Electron Transport in Nanostructures and Mesoscopic Devices: An Introduction.* John Wiley & Sons, 2013.

[35] Franco De Palma. Single-electron transistor on fd-soi for spin qubit sensing, 2021.

[36] J. R. Petta, A. C. Johnson, J. M. Taylor, E. A. Laird, A. Yacoby, M. D. Lukin, C. M. Marcus, M. P. Hanson, and A. C. Gossard. Coherent manipulation of coupled electron spins in semiconductor quantum dots. *Science*, 309(5744):2180–2184, 2005.

[37] Maximilian Russ and Guido Burkard. Three-electron spin qubits. 2016. arXiv preprint arXiv:1611.09106, 28 Nov 2016. Accessed: 2025-08-28.

[38] E. Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Phys. Rev.*, 28:1049–1070, Dec 1926.

[39] Benjamin Schumacher. Quantum coding. *Phys. Rev. A*, 51:2738–2747, Apr 1995.

[40] R. Shankar. *Principles of Quantum Mechanics.* Plenum Press, New York, 2nd edition, 1994.

[41] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[42] F. Simmel, David Abusch-Magder, D. A. Wharam, M. A. Kastner, and J. P. Kotthaus. Statistics of the coulomb-blockade peak spacings of a silicon quantum dot. *Physical Review B*, 59(16):R10441–R10444, 1999.

[43] Nanoacademic Technologies. Lever arm theory, 2025. Accessed: 2025-08-28.

[44] Nanoacademic Technologies. The nonequilibrium green's function (negf) formalism, 2025. Accessed: 2025-08-28.

[45] Nanoacademic Technologies. Transport in the coulomb blockade regime, 2025. Accessed: 2025-08-28.

[46] Nanoacademic Technologies. Tunnel coupling in a double quantum dot in fd-soi. part 1: Plunger gate tuning. Web tutorial, 2025. Accessed: 2025-09-13.

[47] Nanoacademic Technologies. Tutorials - qtcad transport package, 2025. Accessed: 2025-08-28.

[48] The Guardian. Physicist erwin schrödinger's google doodle marks quantum mechanics work. *The Guardian*, August 2013. Accessed: 2025-05-09.

[49] John S. Townsend. *A Modern Approach to Quantum Mechanics.* McGraw-Hill, New York, 1992.

[50] L. A. Tracy, E. P. Nordberg, R. W. Young, C. Borrás Pinilla, H. L. Stalford, G. A. Ten Eyck, K. Eng, K. D. Childs, J. Stevens, M. P. Lilly, M. A. Eriksson, and M. S. Carroll. Double quantum dot with tunable coupling in an enhancement-mode silicon metal-oxide semiconductor device with lateral geometry. *Applied Physics Letters*, 97(19):192110, 2010.

[51] W. G. Van der Wiel, S. De Franceschi, J. M. Elzerman, T. Fujisawa, S. Tarucha, and L. P. Kouwenhoven. Electron transport through double quantum dots. *Reviews of Modern Physics*, 75(1):1–22, 2002.

[52] M. Veldhorst, J. C. C. Hwang, C. H. Yang, A. W. Leenstra, B. de Ronde, J. P. Dehollain, J. T. Muhonen, F. E. Hudson, K. M. Itoh, A. Morello, and A. S. Dzurak. An addressable quantum dot qubit with fault-tolerant control-fidelity. *Nature Nanotechnology*, 9:981–985, 2014.

[53] M. Veldhorst, J. C. C. Hwang, C. H. Yang, A. W. Leenstra, B. de Ronde, J. P. Dehollain, J. T. Muhonen, F. E. Hudson, K. M. Itoh, A. Morello, and A. S. Dzurak. A two-qubit logic gate in silicon. *Nature*, 526:410–414, 2015.

[54] Ke Wang, Hai-Ou Li, Ming Xiao, Gang Cao, and Tang Ping. Spin manipulation in semiconductor quantum dots qubit. *Chinese Physics B*, 27:090308, 09 2018.

[55] Hermann Weyl. *Gruppentheorie und Quantenmechanik.* Hirzel, Leipzig, 1928.

[56] Colin P. Williams. *Explorations in Quantum Computing.* Springer, Berlin, 2011.

[57] Noson S. Yanofsky and Mirco Mannucci. *Quantum Computing for Computer Scientists.* Cambridge University Press, 2013.

[58] Noson S. Yanofsky and Mirco Mannucci. *Quantum Computing for Computer Scientists.* Cambridge University Press, Cambridge, 2013.

[59] D. M. Zajac, T. M. Hazard, X. Mi, E. Nielsen, and J. R. Petta. Scalable gate architecture for a one-dimensional array of semiconductor spin qubits. *Physical Review Applied*, 6(5):054013, 2016.

[60] F. A. Zwanenburg, A. S. Dzurak, A. Morello, M. Y. Simmons, L. C. L. Hollenberg, D. N. Jamieson, J. C. Inkson, S. Rogge, S. N. Coppersmith, and M. A. Eriksson. Silicon quantum electronics. *Reviews of Modern Physics*, 85(3):961–1019, 2013.