Chapter 1

Introduction

1.1 Preface

It is believed that quantum computers will help to increase our knowledge about large molecules and strongly correlated materials. There are many systems in those classes that we do not understand to this day, not for an inability to comprehend their physics, but because we lack computational power. In fact, we have accurate models describing the interactions of electrons in molecules, but classical computers are incompatible in dealing with the quantum properties. While we can easily formulate the Hamiltonians describing the systems, their spectrum and eigenstates (in particular, their ground state), that would allow us insight into their nature, can only be obtained with tremendous efforts. The issue is that with electrons being quantum-mechanical particles, their eigenstates are superpositions of all their possible configurations within the host system. As a consequence, any computer would need to have enough memory to encode the entirety of the electronic Hilbert space. Since the number of configurations is typically exponential in the size of the considered system, the same scaling applies to the resources required for classical simulation. This is a prohibitive overhead that not only places a cap on the number of active electrons we are able to simulate, but also on the simulation accuracy, since increasing the basis set is a means to make a simulation more precise. Fortunately, electronic systems can be stored differently on quantum computers, and so Abrams, Lloyd and Feynman suggested their use as universal quantum simulators [1–3].
A quantum computer is a device with a controllable quantum system as a

memory. As such, the memory spans an exponential Hilbert space which can imitate the exponential Hilbert space of an arbitrary electronic system. The required memory, which is the minimum number of qubits in the quantum computer, is thus only linear in the size of the problem. Not only are quantum states stored more adequately in qubits, but their manipulation by quantum gates, the elementary operations of the quantum computer, allows simulating arbitrary quantum-mechanical time evolutions. This approach, in which quantities of interest are calculated using gate operations rather than only the time evolution of the parent system, is commonly referred to as digital quantum simulation. While being technically more challenging than its analogue counterpart, it allows us to employ powerful quantum algorithms for the simulation and is compatible with techniques of quantum error mitigation and correction. However, since quantum devices have stricter limitations than classical computers, the feasibility of these amazing algorithms is determined by the right 'programming' of the quantum computer, which leads us to the subject matter of this thesis. Within these pages, we are going to deal with three limitations of quantum computing hardware in particular.
Firstly, there is the issue of memory size. Being controllable quantum systems, qubits are not easy to fabricate (or find) and must therefore be regarded as a rare commodity. The era of noisy intermediate scale quantum (NISQ) computers, for instance, is said to open with the arrival of devices containing only around fifty qubits [4]. While the amount of classical memory required to store the Hilbert space of those devices is enormous, the amount of qubits is not. Note also that many promising platforms are currently far from this intermediate scale.

Another issue is the qubit lifetime. Since quantum information can only be stored fleetingly before its decoherence, feasible algorithms have to be short. Quantum computation is thus strongly dependent on our ability to reduce the runtime of quantum algorithms by applying different parts of them in parallel. Not only has the need for parallelization an impact on algorithm design and compilation, but also on the device itself. Quantum algorithms are often expressed as quantum circuit diagrams, in which gate sequences can be grouped into parallel layers, but there is no guarantee that the parallel execution of such a layer is actually possible on the hardware level. Even in fault tolerant quantum computers, where the effect of decoherence is often argued to be diminished,

it never really vanishes. These devices, that are further developed than NISQ computers, continuously run cycles of error correction algorithms that would allow for longer algorithms and so seemingly relax the need for a parallel operation. However, since the cycles are not exempt from noise, they also benefit from parallelization. In fact, when error correction cycles cannot be run within a certain time span, they cause errors rather than correct them.

Lastly, a quantum device has much stricter geometric constraints. This is due to the fact that the actual qubit part of the quantum computer has to be placed inside a dilution refrigerator shielding it from thermal fluctuations. Geometrical constraints not only limit the number of qubits, but also their connectivity and control.

Running a simulation algorithm, all of these peculiarities of quantum hardware need to be taken into account. Let us take a look on how these constraints can enter and influence a computation. Quantum computing is conceptually different from its classical counterpart, even though there are similarities in the way they are applied: for a classical computation, a user would pass problem-specific arguments to pre-existing, standard routines to obtain a result. When concerned with a ground state problem, for instance, a user would initially choose a matrix representation of the Hamiltonian in order to pass it to a diagonalization routine.

The situation is quite similar for digital quantum simulation, where algorithms like *quantum phase estimation* or the *variational quantum eigensolver* [5–7] can be regarded as predefined routines a user has to feed with problem-specific input. Since the problem is finding the ground state the input is, like in the classical case, a representation of the Hamiltonian. It can however not be a matrix, since its sheer dimension would not make it through classical preprocessing. Instead, the Hamiltonian is submitted in a form that the quantum computer can process, as a sum of operators acting on the exponential Hilbert space. To run quantum algorithms, the computer must be able to implement those operations as black box routines on its qubits and they therefore take the form of low-level quantum gates. Typically, the input Hamiltonian is a sum of weighted Pauli strings, where each of them is a product of Pauli operators on different qubits with a real coefficient. Note that with this construction the quantity preventing us from simulating arbitrarily-sized systems is no longer the matrix dimension, but the number of terms in the qubit Hamilto-

nian. The Pauli strings and their associated coefficients are subsequently turned into gate instructions in the quantum algorithm approximating the ground state. Algorithms based on *Trotterization* or *qubitization*, for instance, require that Pauli strings of the Hamiltonian are applied to the system directly [8–10], while in variational quantum eigensolvers they are measured on the memory. In this way, the input Hamiltonian will determine the performance of the applied algorithm, and the number of qubits required for the computation. In order to gain control over those quantities, an amount of pre-processing is required. A visualization of the entire quantum simulation process can be found in Figure 1.1, starting from a fermionic Hamiltonian in second quantization. Considerations about the basis set and truncation of the problem are inherent in this operator, which is to be regarded as a fixed quantity. However, as long as it is a valid representation of the fermionic Hamiltonian, the qubit Hamiltonian can be chosen freely. With that choice, we can influence the performance and requirements of the quantum algorithm that the qubit Hamiltonian feeds into. The transform between the fermionic operators and gate instructions, as well as the correspondence between fermionic occupations and qubit configurations, will be referred to as *fermion-to-qubit mapping*. Since a mapping directly relates the fermionic problem to the computer's memory, it would ideally be tailored to its device. The transform of the Hamiltonians, depicted in Figure 1.1, can be done using classical software readily available [11–13].

1.2 Fermion-to-qubit mappings

Definitions – Qubit basis and Pauli operators

§1 *The state of a single qubit shall be a linear combination of the basis states $|0\rangle$ and $|1\rangle$, which we will also refer to as computational basis. The (single-qubit) Pauli operators X, Y and Z shall be defined such that they are hermitian and unitary.*

$$Z = |0\rangle\langle 0| - |1\rangle\langle 1| \tag{1.1}$$

$$X = |1\rangle\langle 0| + |0\rangle\langle 1| \tag{1.2}$$

$$Y = i\,|1\rangle\langle 0| - i\,|0\rangle\langle 1| \tag{1.3}$$

§2 *To characterize the basis of n qubits, we are going to introduce binary vectors $\boldsymbol{\omega} = (\omega_1, \omega_2, \ldots, \omega_n)^\top$, where each component ω_i is a binary number:*

$\omega_i \in \{0,1\} =: \mathbb{Z}_2$. We thus write $\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}$. For convenience, we are going to use a binary (modular) addition when adding two such binary vectors: $\boldsymbol{\omega} + \boldsymbol{\mu} = \bigotimes_{i=1}^{n} (\omega_i + \mu_i \bmod 2)$. The modulus shall be implicitly applied also to matrix and scalar products of binary vectors. Within this thesis we use those vectors to define the multi-qubit computational basis states as

$$|\boldsymbol{\omega}\rangle \;=\; \bigotimes_{i=1}^{n} |\omega_i\rangle \;=\; |\omega_1\rangle \otimes |\omega_2\rangle \otimes \cdots \otimes |\omega_n\rangle \;, \tag{1.4}$$

such that an n-qubit quantum state $|\varphi\rangle$ takes the generic form

$$|\varphi\rangle \;=\; \sum_{\boldsymbol{\omega} \in \mathbb{Z}_2^{\otimes n}} a_{\boldsymbol{\omega}} \, |\boldsymbol{\omega}\rangle \;, \tag{1.5}$$

where there are 2^n complex, normalized coefficients $a_{\boldsymbol{\omega}}$: $\sum_{\boldsymbol{\omega}} |a_{\boldsymbol{\omega}}|^2 = 1$.

§3 *Products of Pauli operators will be referred to as Pauli strings. To distinguish operators* (1.1)-(1.3) *acting on different qubits, we brand them with qubit indices. The operator X_i for instance acts as an X-operator on qubit i, and as the identity ($\mathbb{I}$) on the rest. In general, we want to omit single-qubit identities and use the following shorthand for Pauli strings $Z \otimes \mathbb{I} \otimes Z = Z_1 \otimes Z_3 = \bigotimes_{i \in \{1,3\}} Z_i$, though we will use $\mathbb{I}$ for the identity operation on the entire system. A qubit Hamiltonian is typically a weighted sum of Pauli strings. In the second chapter, for instance, we find the following Hamiltonian for the hydrogen molecule in a minimal basis (for atoms at their bond distance, in units of Hartree):*

$$\begin{aligned} H \;=\; &-0.34\,\mathbb{I} \;+\; 0.18129\, X_1 \otimes X_2 \;+\; 0.394\, Z_1 \\ &+\; 0.0112\, Z_1 \otimes Z_2 \;+\; 0.394\, Z_2 \,. \end{aligned} \tag{1.6}$$

Definitions – Fermionic operators

§4 *Within this thesis, we denote fermionic creation and annihilation operators by $c_j^\dagger$ and c_j, where j denotes the index of the fermion orbital, the combination of spatial wave function and spin indices. When speaking outside a context of electrons, we also refer to orbitals as fermionic modes, in particular when we replace those indices with coordinates on a hypothetical, two-dimensional embedding of the system. To account for their fermionic*

nature, creation and annihilation operators satisfy the following anticommutation relations:

$$\left[c_i\,, c_j\right]_+ = 0, \quad \left[c_i^\dagger, c_j^\dagger\right]_+ = 0, \quad \left[c_i\,, c_j^\dagger\right]_+ = \delta_{ij}\ , \tag{1.7}$$

where $[A, B]_+ = AB + BA$.

§5 *To avoid confusion with qubit configurations, we will denote the vacuum state, in which all fermion modes (orbitals) are unoccupied, by* $|\Theta\rangle$*. States are subsequently defined as products like* $c_{i_1}^\dagger c_{i_2}^\dagger ... c_{i_M}^\dagger \,|\Theta\rangle$.

§6 *While second quantization makes for some redundancy in the ordering of the creation operators, it allows us to express many-body Hamiltonians in a concise form. The Hamiltonians of the electronic structure problem, that we are interested in, are following the pattern of*

$$\underbrace{\sum_{ij} t_{ij}\ c_i^\dagger c_j}_{\substack{\textit{kinetic / hopping} \\ \textit{and nuclear terms}}} \quad + \quad \underbrace{\sum_{ijkl} V_{ijkl}\ c_i^\dagger c_j^\dagger c_k c_l}_{\textit{two-body interactions}}\ , \tag{1.8}$$

where the sums extend over all mode indices. The coefficients t_{ij} *and* V_{ijkl} *are numbers (integrals depending on the chosen basis) that are generally complex, but related with respect to their indices such that the Hamiltonian is hermitian. Note that by the structure of* (1.8)*, all of its terms conserve the number of particles of a state* $c_{i_1}^\dagger c_{i_2}^\dagger \,...\, c_{i_M}^\dagger \,|\Theta\rangle$*, and most of the terms commute with one another.*

Let us take the qubit requirements as an example of mappings dealing with hardware limitations. The number of qubits storing the problem's Hilbert space is determined by the Hamiltonian: to solve a problem, we need as many qubits as the Hamiltonian acts on. For devices with few (logical) qubits, we are interested in keeping these qubit requirements to a minimum which is determined by the problem's degrees of freedom. Neglecting symmetries like the conservation of particles, mappings typically use more qubits than absolutely necessary. Indeed we need minimally 16 qubits to encode all possible occupations of 16 fermionic modes,

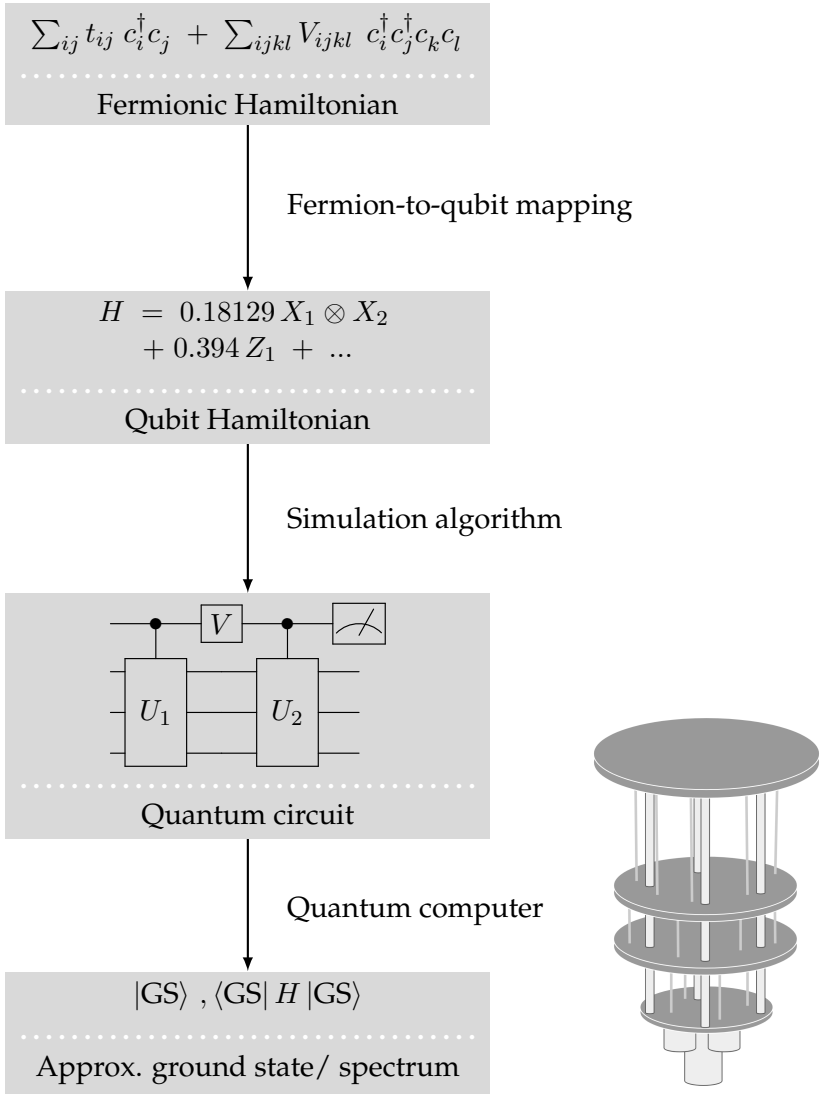

Figure 1.1. Simulating fermions on a quantum computer, depicted as a flow chart. The process starts with a problem Hamiltonian and results in the ground state and its energy being approximated by a quantum computer. However, that ground state problem refers not to the initial Hamiltonian, but rather to a prudently chosen qubit version of it. In combination with the simulation algorithm, the qubit Hamiltonian then determines the quantum circuit, that can be compiled into a program that the quantum computer runs.

but typically not all of those configurations are relevant. Information about the number of fermions in the system is usually given besides the Hamiltonian as either a constraint or from real-life observation of the system. Let us say there is only one particle hopping around in the system given, then the basis of the problem is spanned by 16 configurations with the fermion sitting in a different mode in every basis state. Following combinatorial arguments, these degrees of freedom could be encoded by 4 qubits, and so 12 qubits saved. This of course just an example, but a relevant one since for all particle-number conserving problems, modes are at most half filled (with particles or holes). While encoding all particle numbers usually yields a, in some sense, simpler Hamiltonian, one might be interested in bringing the required number of qubits closer to the minimum. This is the subject of the second chapter of this thesis, in which we consider the impact of classical code layers on fermion-to-qubit mappings. Typically, improvements in the qubit number will have some negative influence on the Hamiltonian. As we will see in the second chapter, this manifests in the choice between either making the gates in the Hamiltonian terms more complex, or accepting a larger number of Pauli string terms instead. Clearly, trading qubits is expensive in the runtime of the simulation algorithm. In fact, one might even consider employing more qubits if only the algorithmic performance could be improved. At least to some degree, this turns out to be possible. While it seems difficult to reduce the number of terms in a Hamiltonian, we can at least make sure they act on the qubit system in a way that would allow us to parallelize the algorithm.
This path is taken in the third chapter, where fermion-to-qubit mappings are enhanced by adding a quantum code layer, for which a number of additional qubits is required. While in the second chapter we have been considering small quantum devices, the focus is now shifted to devices with a larger number of qubits. However we do not want to forgo the aforementioned limitations and assume those devices to contain an unlimited amount of qubits. Instead, let us think of them as being from the NISQ era. With that in mind we will cling to two reasonable strategies. Firstly, we have to compromise between resource requirements and parallelization. With additional qubits to spare, one might for instance be tempted to encode the interactions of the fermionic Hamiltonian directly, but the problem with that is their number, and therefore the number of qubits, usually grows much faster with the system size. Secondly,

due to the connectivity, the quantum device is not going to be able to perform operations between any two qubits. Thus simulated Hamiltonians should ideally only have terms involving coupled qubits, that can be entangled without disturbing others. The quantum codes used in the fermion-to-qubit mappings should thus be defined locally on a realistic layout for a quantum device.
The quantum codes found in the third chapter incorporate both of these strategies: they are local and planar on a square lattice and require a number of qubits that scales with the system size, rather than the Hamiltonian. Note that the square lattice appears to be a natural choice, as it is also the canvas for surface code spurring efforts to build transmon chips in this layout. While we focus on mappings that provide a geometrical embedding of qubit Hamiltonians, their codes would also allow for the application of error mitigation techniques. While those strategies might help improve results in the short term, scalable simulation algorithms are believed to require quantum error correction. The difference between the two is that error mitigation is aiming to filter noise from the obtained data, whereas with error correction, one is aiming to prevent errors during the computation. Practically, quantum error correction codes would constitute a code layer that is unlike the codes in the third chapter. In quantum error correction, physical gate instructions are replaced with their logical counterparts and quantum circuits with their fault-tolerant versions. The entire computation embedded in error correction cycles, in which stabilizers are measured and syndromes are extracted. The problem is that running such cycles is technically challenging even without a computation happening in between.

1.3 Quantum error correction

Definitions – Quantum error correction codes

§7 *A quantum stabilizer code is a mapping between two systems with a different qubit number, where the smaller system is encoded by the larger one. Let us say that the smaller system has n_1 qubits, and the larger one n_2, then a $[[n_2, n_1, d]]$ quantum code maps every state of the former system $|\varphi\rangle$ to a state on the larger system $|\overline{\varphi}\rangle$: $|\varphi\rangle \mapsto |\overline{\varphi}\rangle$. The integer d is called code distance, and will be explained in §10. For the encoding to be a one-*

to-one mapping, these quantum codes constrain $n_2 - n_1$ qubits worth of degrees of freedom by so-called stabilizer conditions. That is, there is a set of commuting Pauli strings such that for each member S we find

$$S\,|\overline{\varphi}\rangle \;=\; |\overline{\varphi}\rangle\;, \tag{1.9}$$

for all encoded states $|\overline{\varphi}\rangle$. S is called a stabilizer. Considering that the identity is part of the stabilizer set, it forms a group generated by $n_2 - n_1$ Pauli strings (using the operator product).

§8 *Not just the states, but also the operators of the smaller system are encoded. For every physical operator $\mathcal{O}$ in the smaller, there exist 'logical' operators $\overline{\mathcal{O}}$ in the larger system, such that*

$$\mathcal{O}\,|\varphi\rangle \;\mapsto\; \overline{\mathcal{O}}\,|\overline{\varphi}\rangle\;. \tag{1.10}$$

Note that there are several logical operators for one physical operator, since their action on the code space is equivalent by the multiplication with stabilizers, i.e. $\overline{\mathcal{O}}$ and $S \cdot \overline{\mathcal{O}}$ have the same effect. Logical operators generally preserve the encoded subspace, i.e. they commute with the stabilizers.

§9 *Stabilizer codes are the workhorse of quantum error correction, since Pauli errors that might occur in a noisy device can either be identified when they anticommute with stabilizers, or are stabilizers themselves and their action therefore trivial. Continuously measuring stabilizers, the system is projected into the subspaces corresponding to outcomes $\langle S\rangle = \pm 1$. Flipped expectation values, referred to as syndromes, can then be attempted to be corrected.*

§10 *With $[[n_2,\, n_1,\, d]]$ error correction codes, one can correct for all conceivable Pauli errors up to a certain weight (the number of nontrivial Pauli operators in the string). Assuming that lower-weight Pauli errors occur with a much higher rate, quantum information is preserved by being stored nonlocally. It is thus unsurprising that the maximal weight of correctable Pauli errors is connected to the minimal weight of logical operators, which is the code distance d. While we can always correct for errors of weight up to $(d-1)/2$, they can be detected up to a weight of $d-1$. In the latter case, the errors that occurred can no longer be discerned, but syndromes may still serve as an indication to discard the outcome of this particular computation. In an error correction setting, Pauli errors with a weight higher than $(d-1)/2$ will generally cause an error on the logical system,*

such that the physical error rate is translated into a logical one. An error correction code is of course only useful if the latter is lower then the former, but there is a threshold of physical errors above which the code causes the rate to increase. Is the noise of a device above this threshold, the goal of fault tolerance, to decrease the logical noise until it becomes negligible, cannot be achieved.

Definitions – Surface code

§11 *An important example of quantum error correction codes is the surface code. The code is popular due to its high threshold [14] and the availability of efficient decoding algorithms [15]. Surface code is the planar version of Kitaev's toric code [16], that in its 'rotated' version [17], is an* $[[d^2, 1, d]]$ *code for an arbitrary odd-valued distance. The* $d = 3$ *and* $d = 5$ *version of the code, as well as the logical operators are depicted in Figure 1.2(b)-(d). Excluded from that count are* $d^2 - 1$ *measurement qubits, that can help perform the syndrome measurements fault-tolerantly, see Figure* 1.2(a). *The code is planar on a square lattice, where the stabilizer generators are overlapping plaquettes that have the structure of* $Z^{\otimes 4}$ *and* $X^{\otimes 4}$*. With the logical operators* $\overline{X}$ *and* $\overline{Z}$ *being defined as* Z*- and* X*-strings from one boundary to the other, the protection of the logical qubit increases with the diameter of the code patch.*

Quantum error correction not only requires many physical qubits, but also precise operations on all of them in parallel. At this point, engineering problems clash with theoretical proposals. In the third chapter we consider two operations parallelizable if they do not use common resources like qubits and couplers (or whatever mediates two-qubit gates), but for real quantum devices, even that is not entirely true. In reality, processes (that for instance implement quantum gates) might not just share quantum resources, but also their control elements: let us say that the qubits are connected to their classical control by some sort of lines. While there has to be a method to select one individual qubit for a quantum gate, it is naïve to assume that a line would not be connected to a number

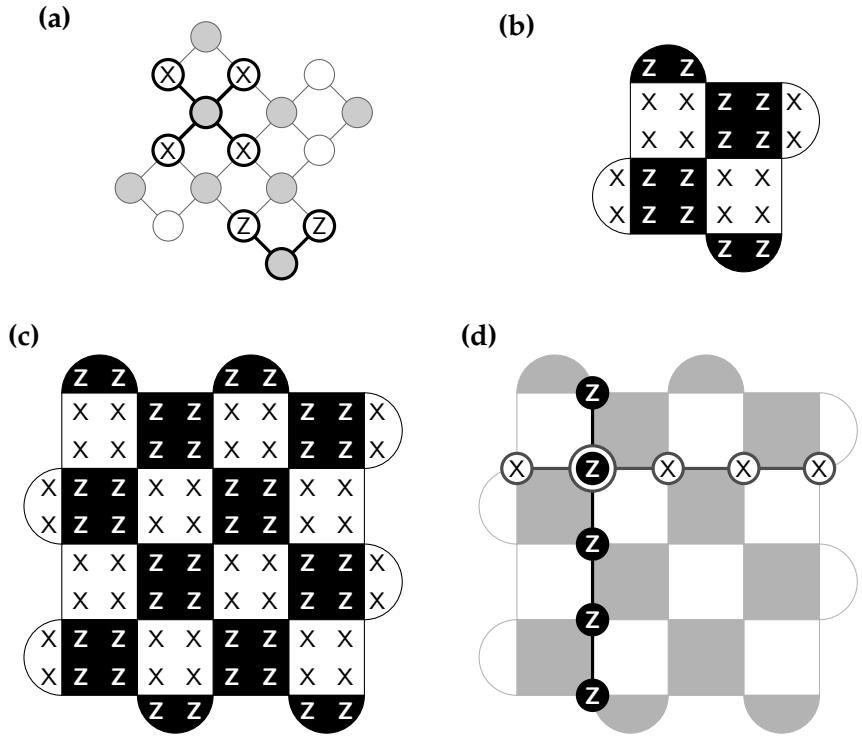

Figure 1.2. Rotated surface code. **(a)** Connectivity graph of the distance-three code. Two stabilizers are highlighted, where for each stabilizer the parity of the involved physical qubits (white), in Z- or X-basis, is collected on the measurement qubit (gray). For that purpose, two-qubit gates have to be performed along the highlighted edges. **(b) & (c)** Stabilizer tiles for the distance three and five code, where every tile is a separate stabilizer with the Pauli operators close to the location of the physical qubits. **(d)** Logical operators of the distance-five code. For rotated surface code, logical operators are Pauli strings across the code patch. $\overline{X}$ and $\overline{Z}$ operators in the figure overlap on exactly one qubit, on which one acts as X, the other as Z such that the logicals anticommute with each other, but commute with all stabilizers.

of qubits at once. Given the sheer amount of qubits, not all of them can possibly be wired individually. In a more realistic scheme, each qubit would have several line contacts and an individual interaction would take place only when multiple of them are operated, see Figure 1.3(a). This allows for individual qubit control, but ultimately has repercussions on parallelization. Imagine two operations in different places where the control lines used for the first and second operation happen to interface at another qubit that is to be left unaffected. Unfortunately, this is only guaranteed as long as the two operations are done sequentially, as parallelizing them will have spurious effects on the qubit at the crossing, shown in Figure 1.3(b). This example is not only far from being unlikely but generalizes into a major drawback of the scheme. However, the reduction of classical control architecture on the quantum device could be regarded as more important. A scalable architecture is a prerequisite for bringing as many qubits as possible onto the same device and into a fridge. As long as those refrigerators do not grow substantially in size over the next years, it will not alone be the amount of control elements to determine whether fault tolerance can be achieved, but also the spatial size of the qubits. The qubit density is of course specific to each platform, and while transmons are a popular technology at the moment, they might eventually be made obsolete by spin qubits in semiconductor quantum dots, for which a much higher density is expected to be achieved [18].

To make use of this density, a crossbar architecture for shared control of spin qubits in silicon quantum dots is proposed in [18]. As we show in the fourth chapter, its limited control mechanisms are sufficient to run quantum error correction cycles without spurious effects. For that purpose we introduce a model of how the quantum dot processor can be controlled from the periphery of the chip. With the focus on bridging the gap between device operations (like control pulses) and quantum circuits, irrelevant details about the device physics are omitted in this model. Taking into account the peculiarities of the system, we discuss parallelization of the available gate operations while providing programmatic steps for the native implementation of surface and color codes. To achieve high fidelity gates, we even restrict the parallelization further. Still, our estimates suggest that a large enough system will be able to suppress the error such that it does not fail more often than a classical memory.

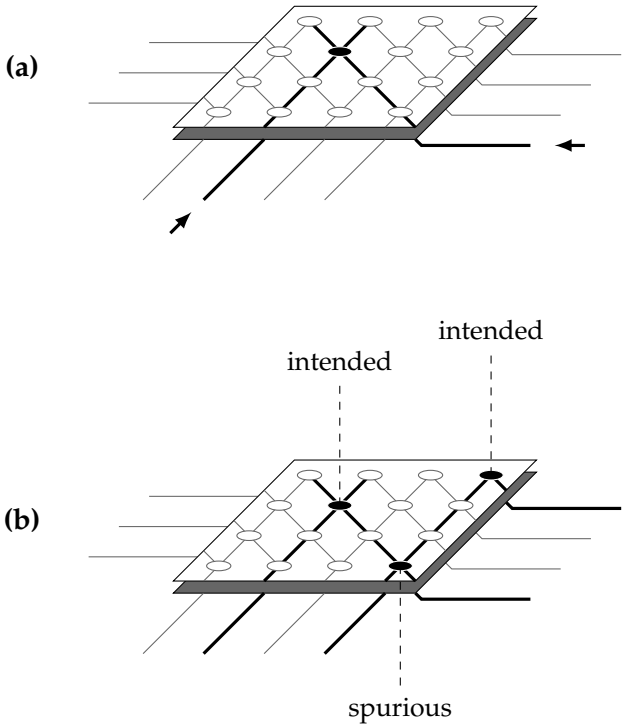

Figure 1.3. Shared control of elements on a chip. **(a)** Each control line connects to several elements. From the periphery, two lines (thickened) are operated to address a single element at their crossing. **(b)** When attempting to address two elements in parallel, the operated lines cross and can cause spurious effects at an unintended crossing.

1.4 This thesis

In this section, we will give a description of the objective in each chapter, and summarize our methods and findings. Each chapter's main contribution is specified. Before we start however, a few words on the structure of this thesis: the individual chapters are opened with a *Background* section and a more detailed discussion about its results. Each chapter's main text is followed by a section called *Supplement* holding additional information referenced in the text, and a table of relevant notations. Scientific progress does not stop while students write their theses, and so a *Further work* section is added to the second and third chapter, describing the latest developments on their respective subjects. This thesis ends with a *Summary* in english and in dutch, the *Curriculum Vitae* of its author and a list of publications.

1.4.1 Chapter two

In the second chapter, we will get in touch with the *Jordan-Wigner transform* [19] in its function as a fermion-to-qubit mapping. It is a simple method, in which each mode is assigned a qubit, indicating fermionic occupation when in the configuration $|1\rangle$. Let us consider the example of Figure 1.4(a), where the modes are represented by small bins and the balls that fill them represent fermions. Beneath the modes, a register of six qubits is shown encoding the state $c_1^\dagger c_2^\dagger c_5^\dagger |\Theta\rangle$. With the Jordan-Wigner transform, the entire Fock space of six orbitals can be encoded in states $|\boldsymbol{\nu}\rangle$ where $\boldsymbol{\nu}$ is any binary vector with, in this case, six components. However, a mere subset of vectors $\boldsymbol{\nu}$ is usually needed to describe the system, which means the Jordan-Wigner transform is inviting unphysical states. In fact, with only the physical states, a number of qubits could be saved. With the resource requirements in mind, we develop a framework for classical encodings of the Jordan-Wigner transform: the idea is to represent only the set of physical configurations $\boldsymbol{\nu}$, but use all possible configurations $|\boldsymbol{\omega}\rangle$ to do so. Encoding fewer degrees of freedom, such a mapping requires fewer qubits. Mathematically, it is defined by an encoding function $e(\boldsymbol{\nu}) = \boldsymbol{\omega}$ and its inverse, the decoding. Since, in particular, the decoding function can be nonlinear with respect to its input $\boldsymbol{\omega}$, the operator transform is re-expressed in terms of a superoperator $\mathfrak{X}$, that can output gate instructions even for those codes. On code examples, we discuss the trade-offs between resource requirements and Hamiltonian size.

The main contribution of this chapter is a general approach for fermion-to-qubit mapping using classical codes.

1.4.2 Chapter three

In the third chapter, we touch upon the subject of the Jordan-Wigner transform of operators. We remark that physical Hamiltonians like (1.8) are build from products $c_i^\dagger c_j$, through which they conserve the number of particles. What terms like $c_i^\dagger c_j$ do is to move fermions from mode j to mode i, so the way to imitate this behavior for the Jordan-Wigner transform is to flip the corresponding qubits. However, since the $c_i^\dagger c_j$ may anticommute with creation operators of $c_{i_1}^\dagger c_{i_2}^\dagger ... c_{i_M}^\dagger |\Theta\rangle$, and so minus signs have to be accounted for. It follows that Pauli strings, translated from hopping terms like $c_i^\dagger c_j$, not only flip qubits i and j, but also scan for anticommutations using Z operators on all qubits in between them. In Figure 1.4(b) it is shown how a fermion hopping from modes 1 to 6 acquires two minus signs from passing particles in modes 2 and 5.
The Pauli string encoding this event is $X_1 \otimes Z_2 \otimes Z_3 \otimes Z_4 \otimes Z_5 \otimes X_6$. Note that this substring of Z-operators does not occur in local hoppings. However, this is a statement about locality in one dimension. In the third chapter, we are interested in harnessing the connectivity of two-dimensional qubit arrays for quantum simulation – a domain in which the Jordan-Wigner transform fails to map local interactions into local gates. A nonlocal string that results from a hopping interaction crossing several rows (when the modes are ordered along a winding pattern from row to row) is shown in Figure 1.4(c). As a one-dimensional fermion-to-qubit mapping, the Jordan-Wigner transform is bound to have nonlocal connections in two dimensions. However, two-dimensionality can be achieved by concatenation with a suitable quantum code. By adding auxiliary qubits and re-constraining them in stabilizers S, an arbitrary state of the memory (1.5) is mapped into something proportional to $[\prod_S (1+S)] |\varphi\rangle \otimes |\chi\rangle$. Since the stabilizers S have an effect on $|\chi\rangle$, the configuration of the auxiliary qubits, one could argue that they store the effect S on $|\varphi\rangle$. By the non-uniqueness of logical operators, we multiply nonlocal strings with stabilizers S, which, if the stabilizers are chosen appropriately, leads to the cancellation of the entire Z-substring. What remains of the logical operator is a local term on the original qubits and a contribution on the auxiliaries. Ensuring these remainders are local is one

of the many tasks covered in the third chapter, in which we also generalize the notion of locality to long-range interactions. The main contribution of this chapter is a novel fermion-to-qubit mapping using quantum codes to manipulate operators.

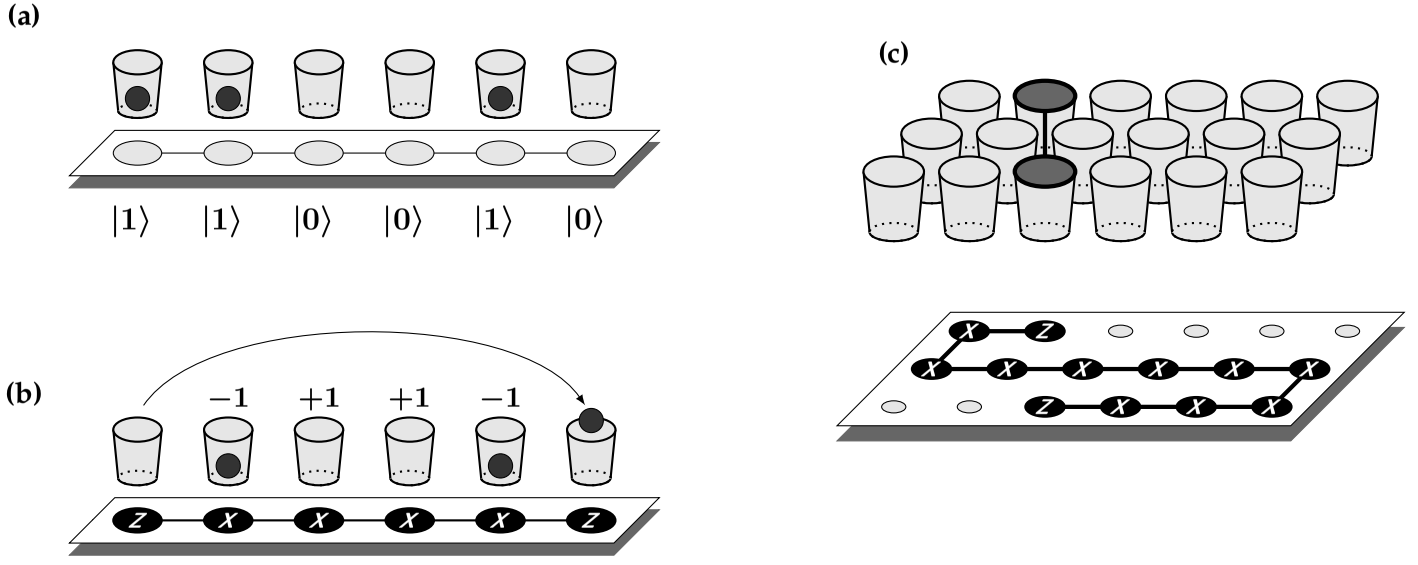

Figure 1.4. Jordan-Wigner transform. The bins represent fermionic modes, the balls fermions. The mapping to qubits on a chip is shown. **(a)** Basis transform with respect to a certain fermionic occupation. **(b)** Operator mapping. An exchange term, in which a fermion changes mode is mapped to a Pauli string in which the corresponding qubits are flipped. The fermion's anticommutations are denoted by '-1' signs and accounted for by Z-operators in the string. **(c)** Nonlocal Pauli strings resulting from a vertical hopping term in an S-pattern version of the Jordan-Wigner transform in two dimensions.

1.4.3 Chapter four

In the fourth chapter, we adapt surface and color code algorithms to the proposed architecture of [18], consisting of a grid of quantum dots on a silicon substrate and physical gates as sketched in Figure 4.1. A quantum dot is a physical site that can confine a single electron serving as a qubit. A square lattice of them is steered by shared control elements contacted from the grid's perimeter. Adjacent dots, for instance, are isolated by barrier gates extending horizontally and vertically over the grid. In this way, one barrier is shared by an entire row or column of dot pairs. To perform a two-qubit quantum gate, the barrier voltage on the barrier between the corresponding dot pairs must be changed, but the operation also affects all parallel dot pairs. This is however not all, since for two-qubit gates, the potential of one of the dots must be changed. The dot potential is manipulated via a type of physical gate, connecting quantum dots diagonally with respect to the direction of the barriers. Only where the diagonal crosses with the barrier, the two-qubit gate is performed. Also, the type and fidelity of the gates varies with the direction the dot pair is aligned. Fortunately, qubit positions can be changed by coherent shuttling [20], such that we can solely rely on high-fidelity gates. The shuttling is instigated by the manipulation of barrier and potentials, such that it can be performed by using the barrier and diagonal gate lines. Shuttling also has a part in performing single-qubit quantum gates at an individual address, since the system only allows to perform these operations globally, on exactly half of the dots. These operations are sufficient to run quantum error correction codes in a highly parallelized fashion, but a higher fidelity is expected when the operation are performed such that parallel two-qubit gates can have individual durations. All of these considerations are considered in the logical error analysis. The main contribution of this chapter is a road map for scalable quantum error correction in silicon quantum dots.