Implementation of quantum algorithms with Josephson charge qubits

Diplomarbeit
von
Norbert Schuch
aus
Temeschburg

durchgeführt am Institut für Physik I – Theoretische Physik
der Universität Regensburg
unter Anleitung von Prof. Dr. Klaus Richter

Dezember 2002

Contents

Chapter 1

Introduction

It was widely believed for a long time that computer science, similar to mathematics, could be treated in a purely abstract way, without considering the underlying physical implementation. In the 1960ies, Landauer [Lan61] and others proved that any physical implementation of computers independent of its concrete realization obeys some general thermodynamical bounds. Namely, each irreversible operation leads to an increase of the entropy which has a lower bound imposed by the laws of thermodynamics. From this, Bennett [Ben73] and others developed a theory of classical reversible computation for which these bounds do not necessarily hold and showed that a classical reversible computer was just as powerful as a conventional one.

Subsequently it became clear that computer science and physics cannot be treated as two independent fields. Since each computation is physical, physical laws impose bounds on the power of computers, but new physical theories might as well allow for new and more powerful ways of computation. In this regard, it is interesting to note that classical computational models work in a way which can be realized fully within the framework of classical physics.

The first idea suggesting that quantum mechanics could allow for more powerful models of computation was given by Feynman [Fey82] who pointed out that the simulation of a quantum mechanical system on a classical computer takes an exponential amount of time, whereas the quantum mechanical system can "simulate" itself in real time. Still, this concept was rather general, and it was not clear at all how quantum mechanics could be employed to really speed up computations.

In 1985, Deutsch [Deu85] proposed a simple computational problem for which a quantum mechanical system (we use the term "quantum computer" from now on) would give an advantage over any classical computer. Although Deutsch's original algorithm had some shortcomings it became evident that there exist tasks where quantum computers can outperform classical ones. Later on Deutsch and Jozsa [DJ92] found an extension to the algorithm giving exponential speed-up over classical computers.

This discovery stimulated the research done in the new field of quantum computation. In the following years, a number of new algorithms were discovered, the most prominent of which are Shor's quantum algorithm for factoring large numbers in polynomial time [Sho94] and Grover's algorithm for searching an unstructured database [Gro96].

These perspectives for practical applications motivated the search for potential implementations of quantum computers. The basic requirement for a quantum computer are controllable quantum mechanical two-level systems which can be coupled in a controlled way. The first experiments were carried out with liquid state NMR since these systems have favourable properties with respect to control and coherence. On the other hand, liquid state NMR has natural restrictions counterstriking the aim to build scalable quantum computers.

Here, similar to classical computers, solid-state implementations seem to be much more promising with respect to scalability. There exists a wide range of proposals how to implement quantum computers using solid state devices, including spins in quantum dots, nuclear spins, and mesoscopic superconducting systems. In this thesis, we will consider a special kind of superconducting quantum bits, the *Josephson charge qubits*.

For many implementations, the Deutsch–Jozsa algorithm has been used as a first demonstration of the feasibility of quantum computation. In 1998, Collins *et al.* [CKH98] showed that the implementation of the Deutsch–Jozsa algorithm in fact could serve as a meaningful test of quantum computation since it makes essential use of the key features of quantum computation, parallelism and entanglement. They introduced a refined version of the algorithm which focuses on these features. Moreover, they could show that only for more than two bits the Deutsch–Jozsa algorithm really involves entanglement. Hence, only up from there it can serve as a meaningful test for quantum computation.

Despite the discovery of more complex quantum algorithms, the Deutsch–Jozsa algorithms still serves as a key test for the feasibility of quantum computation on a given hardware, also because of its relative simplicity. Meaningful (i.e., three-bit) implementations have been given for liquid state NMR and for Josephson junctions coupled by SQUID loops. On the other hand, it turns out that no implementation has been given nor even proposed for more than three qubits. One might think that this is because coherent quantum dynamics is not feasible for more than three qubits. But in fact, quantum algorithms have been implemented for up to seven qubits [VSB$^+$01]. Therefore, it is natural to ask why the existing implementations have not been extended to a higher qubit number.

In this thesis, we show how the Deutsch–Jozsa algorithm can be implemented for more than three qubits. A special focus is put on the implementation using Josephson charge qubits, thus perpetuating the three-bit implementation given by Siewert and Fazio [SF01]. Still, most of our results are not limited to this hardware. In contrast to the existing implementations which employ a classification scheme in order to implement the algorithm, we choose a universal approach.

Thus, we avoid the problems which arise if one tries to extend these schemes, and which seem to be the reason for the lack of implementations for more than three qubits. This universal approach is straightforward to use and enables the implementation of Deutsch's algorithm for all inputs on the same footing. Moreover, it also has applications beyond Deutsch's algorithm which includes the implementation of Grover's algorithm as well as parts of Shor's algorithm.

The work is structured as follows. In Chapter 2, we give an introduction to the basic concepts of quantum computing and explain the Deutsch–Jozsa algorithm. The chapter closes with an overview over some other quantum algorithms.

In Chapter 3, we present the hardware setup of the Josephson charge qubits and show why it can be regarded as a quantum mechanical two-level system. We then discuss various coupling schemes for charge qubits and outline their advantages and disadvantages. The chapter closes with an outlook on other superconducting quantum bits and an overview over experimental results.

Chapter 4 is intended to link the two preceding chapters. We show how the unitary operations introduced in Chapter 2 can be generated using the Hamiltonians obtained in Chapter 3. Additionally, we derive a new operation which is especially well suited for quantum computation with Josephson charge qubits coupled by Josephson junctions. The main results of this chapter were published in [SS02b].

Chapter 5 contains the central results of this thesis. We thoroughly discuss the existing implementations of the Deutsch–Jozsa algorithm for three quantum bits, thus clarifying their underlying principles, and demostrate that an extension of these implementations which bases on the same principles can hardly be accomplished. This leads to the quest for a universal implementation. We therefore introduce the concept of *programmable networks* and give methods how these networks can be implemented on different systems. Thereby, a special focus is put on charge qubits with SQUID coupling. We show how these networks can be used to implement Grover's search algorithm and give some other applications as well. The chapter closes with some considerations on complexity. Preliminary results of this chapter have been reported in [SS02a].

Finally, in Chapter 6 we conclude and give an outlook on some interesting ideas related with our results which could not be treated in this thesis. One of these ideas is given in more detail in the appendix.

As general references, we refer the reader to Preskill's lecture notes [Pre98b] and the book by Nielsen and Chuang [NC00].

Chapter 2

Quantum computation

The purpose of this chapter is to give a brief introduction to quantum computation in general. We introduce basic concepts of quantum computing such as quantum bits, quantum registers, operations, and universality. We also show how quantum computations can be represented by quantum circuits in an easy-to-handle way.

In the second part of this chapter, we try to explain why quantum computers could give an exponential speed-up compared to classical ones. We thoroughly discuss the Deutsch–Jozsa algorithm which will be the most important algorithm for the rest of this work, and finally give a brief overview over some other quantum algorithms.

Especially for this chapter Preskill's lecture notes [Pre98b] and the book by Nielsen and Chuang [NC00] may serve as general references.

2.1 Quantum computers

2.1.1 Quantum bits

Single quantum bits

In classical computation, the basic unit of information is one *bit*, which can have two possible states, 0 and 1. Therefore, if thinking about computation based on the principles of quantum mechanics, it is natural to take the corresponding quantum mechanical system, i.e., a system with two basis states, $|0\rangle$ and $|1\rangle$. This could be a spin, some atom in its ground or an excited state, the polarization of a photon, etc. Such a two-level system is called a *quantum bit* or *qubit*— because of the correspondence to classical computation, and because a two-level system is the simplest quantum mechanical system and therefore the basic unit of information in quantum information science.

Obviously, the first big difference between a classical bit and a quantum bit is the fact that the classical bit can only be in one of the two states 0 and 1,

whereas the qubit also can take any superposition of the two states $|0\rangle$ and $|1\rangle$, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$, $|\alpha|^2 + |\beta|^2 = 1$. Therefore, a quantum bit can obviously store more information[1] than a classical bit. On the other hand, we cannot access the information stored in a quantum bit directly: a quantum mechanical measurement—projecting, e.g., onto the two basis vectors $|0\rangle$ and $|1\rangle$—will only yield one bit of classical information.

As it is common in the description of spins in quantum mechanics, we will use two-component vectors to describe the state of a quantum bit, i.e.,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} .$$

Unless mentioned explicitly, kets labelled with roman letters like $|x\rangle$ denote computational basis states (i.e., $|0\rangle$ or $|1\rangle$), whereas greek letters like $|\psi\rangle$ will be used for arbitrary states of the system. Since quantum mechanics is linear, the action of some mapping is already defined by its action on the basis states.

In order to perform computations it is necessary to manipulate the (qu)bits in a controlled way. Classically, the only nontrivial operation on a single bit is the NOT operation exchanging 0 and 1. In quantum mechanics, at least in principle any arbitrary unitary operation can be applied to the state since the quantum mechanical time evolution is unitary. Of course, whether a certain unitary operation can be applied or not crucially depends on the Hamiltonian of the system. The relation between Hamiltonians and unitaries will be discussed in Chapter 4.

Some of the most important matrices are the Pauli matrices together with the identity matrix,

$$I = \begin{pmatrix} 1 & \\ & 1 \end{pmatrix}; \quad \sigma_x = \begin{pmatrix} & 1 \\ 1 & \end{pmatrix}; \quad \sigma_y = \begin{pmatrix} & -i \\ i & \end{pmatrix}; \quad \sigma_z = \begin{pmatrix} 1 & \\ & -1 \end{pmatrix} .$$

These matrices are unitary as well as hermitian and will be used most often throughout this work. Note that in quantum computing matrices are given in the $\{|0\rangle, |1\rangle\}$ basis, i.e., the eigenvalue with respect to σ_z of the $|0\rangle$ state is 1 while the eigenvalue of the $|1\rangle$ state is -1. The σ_x operation is the NOT operation for qubits, but unlike in classical computation, there also exist infinitely many other nontrivial operations.

Quantum registers

Still, one qubit does not make a quantum computer.[2] Therefore, we combine a number of quantum bits to a *quantum register*, as it is the case in classical

[1] Admittedly, this is put a bit fuzzy; we will not discuss information measures in this work. This is rather meant qualitatively.

[2] This is not perfectly true though. As we will see with Deutsch's algorithm, one qubit can be sufficient to do certain tasks. Nevertheless, we would like to go beyond one qubit.

computation. Quantum mechanics now takes place in the product space spanned by the basis states of the N qubits $|x_1\rangle \otimes |x_2\rangle \otimes \ldots \otimes |x_N\rangle$. This means that our computation takes place in a 2^N-dimensional Hilbert space! Consequently, we can apply any $\mathrm{U}(2^N)$ transformation to the state of our register, at least in principle.

We would like to add some remarks about the notations for N-qubit registers. We will use equivalently the notations $|x_1\rangle \otimes |x_2\rangle \otimes \ldots \otimes |x_N\rangle \equiv |x_1\rangle|x_2\rangle \ldots |x_N\rangle \equiv |x_1 \cdots x_N\rangle \equiv |x\rangle \equiv |\mathbf{x}\rangle$. The states $|x_1 \cdots x_N\rangle$, $x_i = 0, 1$ are represented by N-digit binary numbers (i.e., vectors). The order of the qubits in the ket is chosen such that the first qubit is the leftmost, i.e., the most significant digit. In case the register contains a binary encoded natural number and not only a binary vector, we also denote the ket by the corresponding number, which then can be read as a binary code, e.g., $|19\rangle$ instead of $|010011\rangle$. This correlates with the use of $|x\rangle$ vs. $|\mathbf{x}\rangle$ for registers which store numbers or binary vectors, respectively. As in the one-qubit case, the states will be ordered ascendingly, e.g., $\{|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, \ldots, |111\rangle\}$.

Entanglement

It is important to note that the step from one qubit to more qubits adds a new resource (compared to classical computation) apart from the possibility of superposing states: entanglement. Quantum mechanical systems composed of some subsystems usually cannot be described simply by giving the states of all the subsystems separately. Some of the information about the state of the system is shared between the subsystems, a phenomenon called entanglement. Entanglement seems to be a key resource in quantum information and quantum computation [JL02]. Still, neither entanglement in general nor its role in the speed-up observed in quantum algorithms is fully understood, although it seems that all quantum algorithms which do not use entanglement can be reformulated such that the speed-up can be obtained on a classical computer, too.

The classic example for an entangled state is the Bell state

$$|\psi\rangle = \frac{1}{\sqrt{2}} \left(|00\rangle + |11\rangle\right) \ .$$

Clearly, this state cannot be written as a product of two one-qubit states,

$$|\psi\rangle \neq \Big[\alpha_A|0\rangle + \beta_A|1\rangle\Big]\Big[\alpha_B|0\rangle + \beta_B|1\rangle\Big] \ .$$

2.1.2 Universal quantum computation

As we have mentioned before, quantum computation requires complex unitary transformations of the quantum registers which themselves can get very large. Assembling each of these unitary transformations directly from the Hamiltonian

of the system (which has natural restrictions) is a tedious work. It would be desirable to have a restricted set of basic building blocks and a prescription how to assemble complex unitaries using these basic operations.

Such sets of operations which in principle allow for the realization of any unitary operation are called *universal sets*. Again, the idea is taken from classical computation: in classical computation, there are basic building blocks which suffice to build any logic operation on any number of bits. (In fact, the so-called NAND gate which takes two inputs x and y and returns NOT(x AND y) is sufficient). In quantum computation, one can try to find such universal gates as well. Since the space of unitary $2^N \times 2^N$ matrices, unlike the space of classical logical operations on N bits, is continuous, one might look for operations which *approximate* any U(2^N) arbitrarily well. Deutsch [Deu89] found that there is a three-qubit matrix which is sufficient for this purpose. Later on, Barenco [Bar95] derived a class of two-bit matrices each of which was universal, too, in the sense mentioned above.

In a seminal work, Barenco *et al.* [BBC$^+$95] could show in 1995 that any unitary operation on N qubits could be generated *exactly* using one special (itself non-universal) two-qubit operation together with the set of all one-bit operations. Moreover, they could also give a couple of recipes how to generate a wide range of important operations using their universal set of operations.

The considered two-bit operation was

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{pmatrix} ,$$

which corresponds to the mapping

$$\begin{array}{lll} |00\rangle \mapsto |00\rangle & , & |10\rangle \mapsto |11\rangle \\ |01\rangle \mapsto |01\rangle & , & |11\rangle \mapsto |10\rangle \end{array} ,$$

of the basis states. It can be seen easily that this operation negates the second qubit exactly if the first one is in the $|1\rangle$ state, and therefore is called the controlled-NOT (CNOT) gate. Alternatively, it is sometimes called the XOR gate, since it effectively maps

$$|x_1, x_2\rangle \mapsto |x_1, x_1 \oplus x_2\rangle ,$$

where $\oplus$ is the logical XOR operation, i.e., the addition modulo 2.[3]

[3] A classical XOR gate has only one output. But since quantum mechanical (i.e., unitary) time evolution is reversible and the classical XOR is not, this operation cannot be implemented on a quantum computer. Actually, the theory of *reversible* classical computation developed in the 1970ies [Ben73] is much more related to quantum computation, but discussing this would lead to far.

It is interesting to note (and could be one of the reasons for the widespread use of CNOT in quantum computation) that the CNOT operation can be viewed as a classical computational operation since it does not mix the basis states nor introduces any phases. Therefore, it can be handled much more intuitively than operations being genuine quantum mechanical. For a proof of the fact that CNOT together with arbitrary one-bit operations is universal we refer to the original work since it is quite lengthy and we will not need that result for the main results of this thesis.

We mention that it has been proven recently that any arbitrary nontrivial (i.e., entangling) two-qubit gate together with the set of all one-bit operations is universal [BB01, BDD$^+$02].

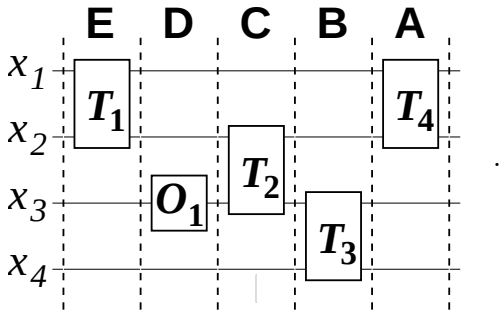
2.1.3 Quantum circuits

As mentioned in the last subsection, unitary transformations in quantum computing will usually be built using one- and two-bit operations. This means that the $2^N \times 2^N$-matrix of the N-qubit operation can be written as a product of unitary operations where each factor is an operation on one or two qubits, i.e., a direct (tensor) product of the operation with the identity operator on all other qubits.

Consider the following (hypothetical) example for four qubits, where the O_i are one-bit operations and the T_i are two-bit operations, I being the one-qubit identity matrix:

$$\underbrace{T_4 \otimes I \otimes I}_{\mathsf{A}} \cdot \underbrace{I \otimes I \otimes T_3}_{\mathsf{B}} \cdot \underbrace{I \otimes T_2 \otimes I}_{\mathsf{C}} \cdot \underbrace{I \otimes I \otimes O_1 \otimes I}_{\mathsf{D}} \cdot \underbrace{T_1 \otimes I \otimes I}_{\mathsf{E}}$$

and compare it with the following diagram, where qubits are denoted by lines and unitary operations by symbols on the lines they act on (in this case boxes):



Note that the operations in quantum circuits (that's how these diagrams are usually termed) are placed from left to right as they act on the initial state, which means that they appear in the reverse order compared to the matrix notation.

The notion of quantum circuits has several advantages: not only they are easier to read, they also make some properties of the unitary operations in product

space more obvious. For instance, $I \otimes I \otimes O_1 \otimes I \cdot T_1 \otimes I \otimes I$ (i.e., the blocks D and E) can be collected to $T_1 \otimes O_1 \otimes I$. This can be seen directly in the circuit notation, as well as the fact that both operations therefore commute. Furthermore, circuits make it easier to denote operations between distant qubits (although *boxes* might be a bit unappropriate for that purpose).

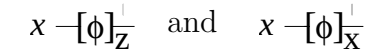
Most of the symbols used in quantum circuits will be introduced when needed. Single-qubit operations are often denoted by boxes with the name of the corresponding matrix inside. Two special classes of one-bit operations are the rotations around the z axis, which are obtained by switching on a σ_z Hamiltonian

$$R_z(\phi) = e^{-i\sigma_z \phi/2} = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix} \tag{2.1}$$

and the x rotations (obtained by switching on a σ_x Hamiltonian):

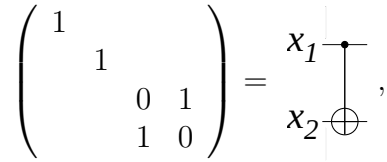
$$R_x(\phi) = e^{-i\sigma_x \phi/2} = \begin{pmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{pmatrix} . \tag{2.2}$$

In quantum circuits, we denote these operations by

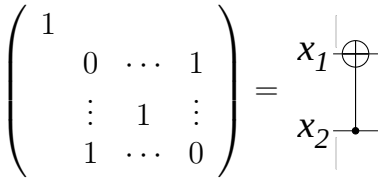


respectively.

Finally, the symbol for the CNOT gate is



where the qubit marked with $\oplus$ (the second one) is the one being negated and the qubit marked with a dot is the one controlling the negation. Of course, there also exists the other version of CNOT, where the second qubit is the control bit:



2.1.4 Error correction

Much more than in classical computation, errors are a very serious issue in quantum computation. Unlike in classical computation, where a bit flip is the only possible error, quantum mechanical states can deviate continuously from their nominal value. They lose their coherence by interaction (entanglement) with the

environment. This effect is called decoherence. We will not discuss decoherence here; further reading can be found, e.g., in [PZ99].

Luckily, there exist possibilities to correct these errors. The main idea of *Quantum Error Correction* is to encode the two states $|0\rangle$ and $|1\rangle$ of a single qubit into two orthogonal states of k qubits, where k depends on the code chosen. The two encoding states are entangled states of the k qubits. Thus, errors occuring only locally on one or a few qubits (depending on the code!) can be detected by indirectly measuring the qubits. By the measurement, small perturbations of a single qubit are either projected back to the original state or extended to a spin flip or something similar. The outcome of the measurement allows us to discriminate the two cases and to apply correction operations when appropriate. See [Pre98a], [NC00] or [KW02] for further reading on this.

Still, it should be noted that in order to get the error correction working, the qubits have to fulfill at least some minimum requirements. Especially, the ratio between the time $\tau_{\text{op}} = \hbar/E$ typically needed for one-bit operations (where E is the typical eigenvalue of the qubit Hamiltonian) and the decoherence time τ_{dec} has to be at least of order of magnitude 10 000 (see, e.g., [Pre98a] or [NC00]). Consequentially, this ratio is a key requirement for a good qubit implementation.

2.2 Quantum algorithms

The second part of this chapter contains an introduction to quantum algorithms. A special focus is put on the Deutsch–Jozsa algorithm, since it is the main topic of this thesis.

2.2.1 The Deutsch–Jozsa algorithm

Deutsch's problem

Take a Boolean function $f : \{0,1\} \to \{0,1\}$ on one bit. There are four possible functions of this kind, namely

$$\begin{aligned} f_0 &\equiv (0,0)\,, \\ f_1 &\equiv (0,1)\,, \\ f_2 &\equiv (1,0)\,, \\ \text{and} \quad f_3 &\equiv (1,1)\,. \end{aligned}$$

where we denote each f by the tuple $(f(0), f(1))$.

Obviously, two of the functions (f_0 and f_3) are constant while the other two are not. Now imagine somebody prepared a black box where you could insert a Boolean value x and the box would tell you about $f(x)$ (see Fig. 2.1a). Your task is to find out whether the f in the black box is constant or not. Obviously, to solve this task you would need two queries to this black box.

Now the interesting thing is that a quantum computer can accomplish this task with only *one* query to the black box (which, from now on, we will often call the *oracle*, since it tells us about f).

Quantum oracles

In order to understand why—and how—this works, we have to do some preliminary work. (Well, we do not have to do this necessarily, but things will hopefully seem less counter-intuitive that way.) In order to keep tasks comparable, first of all we have to find out how to build black boxes which can be used in quantum as well as in classical computers in an at least analogous way. Of course, we simply could try to take the old black box from Fig. 2.1a and extend it such that it not only takes bits but qubits as inputs and outputs, thus mapping $\alpha|0\rangle + \beta|1\rangle$ to $\alpha|f(0)\rangle + \beta|f(1)\rangle$ (see Fig. 2.1b). For input states $|0\rangle$ and $|1\rangle$, this would include the classical case, but it would as well allow for a quantum mechanical evaluation of our black box. But, as a matter of fact, this mapping is not reversible, at least not for constant f—whereas each quantum mechanical evolution has to be reversible. Therefore, we have to look for a—simultaneously classical and quantum mechanical—implementation of our black box *which is reversible*.

A first step towards reversibility would be to additionally output the inserted value x (see Fig. 2.2a). Furthermore, since reversibility clearly requires the conservation of the number of (qu)bits (non-quadratic matrices are never invertible), we have to provide another state, which we initialize to 0 and which will be set to $f(x)$ by the black box (Fig. 2.2b). Finally, since the circuit must be reversible in any case and not only in the case where the extra line is initialized to 0, we have to find a way to make the oracle reversible for any initial value of the second bit. In order to accomplish this, we can *invert* the second bit exactly if $f(x)$ is one: this gives $f(x)$ if $y = 0$ and also preserves reversibility for $y = 1$. Mathematically, this corresponds to mapping y to $y \oplus f(x)$ —a controlled-NOT operation![4]

This implementation of our black box can now be transferred one-to-one to a

[4]... although the control condition is not necessarily the state of the first qubit. It is therefore a more general kind of 'control'.

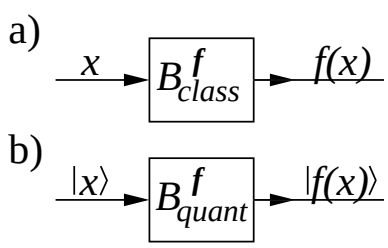


Figure 2.1: a) Simple black box model for the implementation of an one-bit Boolean function f on a classical computer. **b)** One-to-one transfer of a) to a quantum computer, where $|x\rangle$ is $|0\rangle$ or $|1\rangle$, thus defining the mapping on the basis. Still, for certain fs the mapping is not reversible and therefore not quantum mechanical.

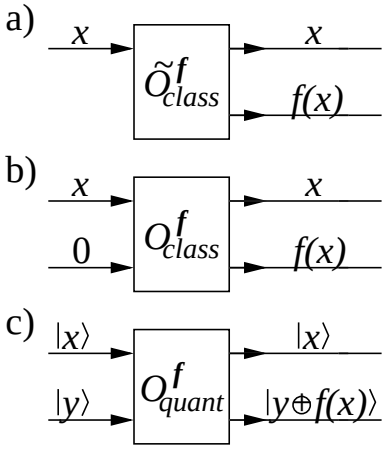


Figure 2.2: Steps towards a reversible black box for Boolean functions f. **a)** Black box which returns the input, thus yielding reversibility in principle. **b)** Since bits cannot be generated, a second input bit has to be added, which returns $f(x)$. **c)** The second input bit has to warrant reversibility as well. The resulting oracle c) can also be used in quantum circuits.

quantum mechanical setup (Fig. 2.2c) and still remains reversible. We can use it as a classical oracle if restricting the inputs to the 'classical' states $|0\rangle$ and $|1\rangle$, but we can as well insert any superposition.

To reconsider Deutsch's problem, it is clear that also in this setup two classical queries to the oracle are needed in order to find out whether f is constant or not.

The Deutsch algorithm

In the following, we present the Deutsch algorithm which solves Deutsch's problem with only one query. Nevertheless, we do not present Deutsch's original algorithm [Deu85] but the refined version by Cleve *et al.* [CEMM98]. The same applies to the subsequent section about the Deutsch–Jozsa algorithm [DJ92].[5]

Consider the following quantum mechanical evaluation of f:

1. Initialize the first qubit ($|x\rangle$) to $|0\rangle$ and the second one to $|1\rangle$.

2. Apply a Hadamard transform *to each of the two qubits*. The Hadamard transform is defined as

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} .$$

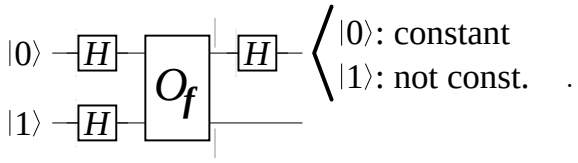
3. Apply the oracle O_f.

4. Apply another Hadamard transform H *to the first qubit only*.

5. Measure the first qubit in the computational basis $\{|0\rangle, |1\rangle\}$. If the system is found in the $|0\rangle$ state, the function implemented in the oracle is constant. In case the outcome is $|1\rangle$, it is not.

[5]Deutsch's original algorithm only succeeded in half of the cases. Still, it returned a bit telling whether it succeeded or not.

The whole procedure can be illustrated by the following quantum circuit:



Before discussing some of the astonishing aspects of quantum algorithms which can already be seen in this simple example, let us first check that the algorithm really works. We will describe the state of our system as a product of two kets, where the first ket represents the upper and the second ket the lower qubit.

At the beginning, the register of our quantum computer is in the state

$$|\psi_{\text{initial}}\rangle = |0\rangle|1\rangle \ .$$

The first Hadamard block maps $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$:

$$\begin{aligned} |\psi_{\text{before\_}O_f}\rangle &= \frac{1}{2}\Big[|0\rangle + |1\rangle\Big]\Big[|0\rangle - |1\rangle\Big] \\ &= \frac{1}{2}\Big[|0\rangle\Big(|0\rangle - |1\rangle\Big) + |1\rangle\Big(|0\rangle - |1\rangle\Big)\Big] \ . \end{aligned}$$

Then we apply the oracle. This maps each $|x\rangle|y\rangle$ to $|x\rangle|y \oplus f(x)\rangle$, thus yielding

$$|\psi_{\text{after\_}O_f}\rangle = \frac{1}{2}\Big[|0\rangle\Big(|0 \oplus f(0)\rangle - |1 \oplus f(0)\rangle\Big) + |1\rangle\Big(|0 \oplus f(1)\rangle - |1 \oplus f(1)\rangle\Big)\Big] . \tag{2.3}$$

We now have two expressions of the form

$$|0 \oplus a\rangle - |1 \oplus a\rangle \ , \tag{2.4}$$

where $a \in \{f(0), f(1)\}$ is a Boolean value. So in case $a = 0$, the expression (2.4) is

$$|0\rangle - |1\rangle = (-1)^a\Big(|0\rangle - |1\rangle\Big)$$

and in case $a = 1$ (with $1 \oplus 1 = 0$)

$$|1\rangle - |0\rangle = (-1)^a\Big(|0\rangle - |1\rangle\Big) \ .$$

By inserting this into (2.3), we obtain:

$$\begin{aligned} |\psi_{\text{after\_}O_f}\rangle &= \frac{1}{2}\Big[|0\rangle(-1)^{f(0)}\Big(|0\rangle - |1\rangle\Big) + |1\rangle(-1)^{f(1)}\Big(|0\rangle - |1\rangle\Big)\Big] \\ &= \frac{1}{\sqrt{2}}\Big[(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\Big]\frac{1}{\sqrt{2}}\Big[|0\rangle - |1\rangle\Big] \ . \end{aligned}$$

Finally, we apply the second Hadamard transform H to the first qubit, which yields the final state

$$|\psi_{\text{final}}\rangle = \frac{1}{2}\left[\Big((-1)^{f(0)} + (-1)^{f(1)}\Big)|0\rangle + \Big((-1)^{f(0)} - (-1)^{f(1)}\Big)|1\rangle\right]\frac{1}{\sqrt{2}}\Big[|0\rangle - |1\rangle\Big] ,$$

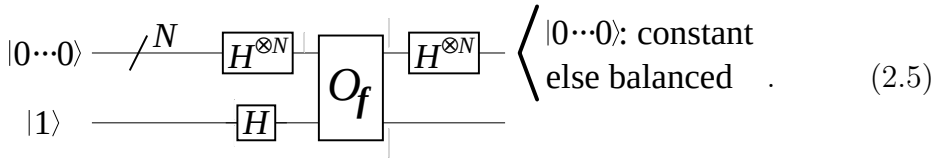
and one immediately sees that for f constant the amplitude for finding the first qubit in the $|1\rangle$ state cancels out, while for f non-constant, the amplitude of $|0\rangle$ vanishes. Therefore, if f is constant we measure $|0\rangle$ in the first register with certainty, otherwise $|1\rangle$.

The Deutsch–Jozsa algorithm

This algorithm actually does not only work for a function f defined on one qubit, but for an arbitrary number N of qubits. Take a function $f : \{0,1\}^N \to \{0,1\}$ where it is promised that f is either constant or balanced. Here, balanced means that the number of zeros and ones as outcomes of f is equal. (In the one-bit case this is true for all non-constant functions.) In this case, the Deutsch–Jozsa algorithm can tell us with *one* evaluation of f whether the function is constant or balanced. Classically, this would require $2^{N-1} + 1$ queries to the oracle in the worst case.

The N-qubit Deutsch–Jozsa algorithm works exactly the same way as the one-bit version does, except that we replace the upper line by an N-qubit register $|x_1, \ldots, x_N\rangle \equiv |\mathbf{x}\rangle$, so that the oracle now acts on $N+1$ qubits as $O_f : |\mathbf{x}\rangle|y\rangle \mapsto |\mathbf{x}\rangle|y \oplus f(\mathbf{x})\rangle$.

The N-qubit Deutsch–Jozsa algorithm is represented by the following circuit:



(2.5)

Here, the upper line marked with a bar and 'N' denotes an N-qubit register. $H^{\otimes N}$ denotes the application of the Hadamard transform on each qubit separately, i.e., the outer product of N Hadamard transforms. Most often, we call it the N-bit Hadamard transform, $H_N \equiv H^{\otimes N}$. If unambiguous, we will sometimes even denote it simply by H.

In order to show how the N-bit version of the Deutsch–Jozsa algorithm works, we start with an alternative description of H_N. The one-bit Hadamard transform can be rewritten as

$$H : |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y\in\{0,1\}} (-1)^{xy}|y\rangle .$$

Therefore, the outer product of N Hadamard transforms can be expressed as

$$H_N : |x_1, \ldots, x_N\rangle \mapsto \frac{1}{\sqrt{2^N}} \sum_{y_1,\ldots,y_N} (-1)^{x_1y_1+\ldots+x_Ny_N}|y_1, \ldots, y_N\rangle .$$

With the abbreviation $|\mathbf{x}\rangle \equiv |x_1, \ldots, x_N\rangle$, and by taking into account that the sum in the exponent can also be taken modulo two, therefore defining $\mathbf{x} \cdot \mathbf{y} = x_1 y_1 \oplus \ldots \oplus x_N y_N$, we can rewrite this as

$$H_N : |\mathbf{x}\rangle \mapsto \frac{1}{\sqrt{2^N}} \sum_{\mathbf{y}} (-1)^{\mathbf{x}\cdot\mathbf{y}} |\mathbf{y}\rangle \;, \tag{2.6}$$

where the sum is taken over all possible N-qubit basis states $\mathbf{y}$.[6]

Having understood the N-qubit Hadamard transform—which is a product of one-bit operations!—we start to check the Deutsch–Jozsa algorithm, as given in (2.5):

$$\begin{aligned}
|0\cdots 0\rangle\ |1\rangle \quad &\overset{H^{\otimes N}\otimes H}{\longmapsto} \quad \frac{1}{\sqrt{2^N}} \left[\sum_{\mathbf{x}} (-1)^{\mathbf{0}\cdot\mathbf{x}} |\mathbf{x}\rangle \right] \ \frac{1}{\sqrt{2}} \Big[|0\rangle - |1\rangle \Big] \\
&= \frac{1}{\sqrt{2^N}} \sum_{\mathbf{x}} |\mathbf{x}\rangle \ \frac{1}{\sqrt{2}} \Big[|0\rangle - |1\rangle \Big] \\
&\overset{O_f}{\longmapsto} \quad \frac{1}{\sqrt{2^N}} \sum_{\mathbf{x}} |\mathbf{x}\rangle \ \frac{1}{\sqrt{2}} \Big[|0 \oplus f(\mathbf{x})\rangle - |1 \oplus f(\mathbf{x})\rangle \Big] \\
&= \frac{1}{\sqrt{2^N}} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle \ \frac{1}{\sqrt{2}} \Big[|0\rangle - |1\rangle \Big] \\
&\overset{H^{\otimes N}\otimes I}{\longmapsto} \quad \frac{1}{2^N} \left[\sum_{\mathbf{y}} \sum_{\mathbf{x}} (-1)^{\mathbf{x}\cdot\mathbf{y}} (-1)^{f(\mathbf{x})} |\mathbf{y}\rangle \right] \ \frac{1}{\sqrt{2}} \Big[|0\rangle - |1\rangle \Big] \;.
\end{aligned} \tag{2.7}$$

We now measure the first register. The probability amplitude for the $|0\cdots 0\rangle$ state is

$$\left| \frac{1}{2^N} \sum_{\mathbf{x}} (-1)^{\mathbf{x}\cdot\mathbf{0}} (-1)^{f(\mathbf{x})} \right|^2 = \left| \frac{1}{2^N} \sum_{\mathbf{x}} (-1)^{f(\mathbf{x})} \right|^2 \;,$$

which obviously equals one if f is constant and vanishes if f is balanced. Therefore, whether the first register is in the zero state or not is a sufficient criterion to distinguish constant and balanced functions f, so that the Deutsch–Jozsa algorithm works.

The Collins–Kim–Holton version of the Deutsch–Jozsa algorithm

It is interesting to note that, after the evaluation of the oracle, we do no longer care about the last qubit—which in the classical case is the *only* line carrying information about the function f ! In fact, the last qubit is in the same state after the application of the oracle as before. Particularly, the last qubit does

[6] The operation $\mathbf{x} \cdot \mathbf{y}$ is sometimes called the 'outer product' or even the 'scalar product' of $\mathbf{x}$ and $\mathbf{y}$, although it is not a scalar product.

not get entangled with the other ones. This is indeed very surprising.[7] The first to point this out with the Deutsch–Jozsa algorithm were Collins, Kim, and Holton [CKH98]. From that, they concluded that the last ("auxiliary") qubit was actually not necessary for the speed-up in the Deutsch–Jozsa algorithm and therefore could be omitted. To this end, they suggested the use of an "optimized" oracle U_f acting only on the remaining N qubits. On these bits, it should work the same way the old oracle did in the presence of the auxiliary qubit (see (2.7)), i.e.,

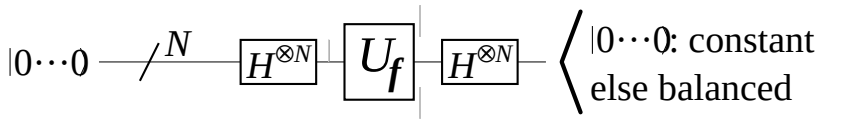
$$\sum_{\mathbf{x}} \alpha_{\mathbf{x}} |\mathbf{x}\rangle \stackrel{U_f}{\longmapsto} \sum_{\mathbf{x}} \alpha_{\mathbf{x}} (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle \ .$$

Directly speaking, U_f flips the phase of each basis state $|\mathbf{x}\rangle$ conditionally upon $f(\mathbf{x})$. This means that it acts on the computational basis as

$$U_f : |\mathbf{x}\rangle \mapsto (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle \ ,$$

i.e., as a diagonal matrix with entries $(-1)^{f(\mathbf{x})}$. This type of oracle is sometimes called an "f-controlled phase flip". In contrast to the classical oracle O_f, we denote it by U_f.

By using this type of oracle, we can perform the Deutsch–Josza algorithm exactly the way we did before, except that the auxiliary qubit can be omitted:



The proof goes analogously except that the ancilla (a short name for an auxiliary qubit) is missing.

Based on this, Collins, Kim, and Holton conclude that it actually is sufficient to state the Deutsch problem with the reduced type of oracle: the exponential speed-up in terms of oracle queries is still there, and the algorithm gets more simple—its essentials become more clear, so to say. They especially point out that in case someone wants to implement the whole Deutsch–Josza algorithm, including the oracle, in order to show the suitability of a certain hardware setup as a quantum computer, it is sufficient to realize this reduced version of the algorithm. They argue that the main resource which has to be demonstrated in this case is the ability of the system to handle entanglement, and therefore the ancilla is not necessary for a proof of the feasibility of quantum computation on the tested hardware.

Following the argument that entanglement is the key point to be demonstrated by an implementation of the Deutsch–Jozsa algorithm, they also point out that U_f actually does not involve entanglement at all for one or two qubits. For one

[7] Note that this does not mean that the oracle cannot entangle the last qubit with the others. It is simply a matter of the initial state of the last qubit.

qubit, this is clear, and also for two qubits it can be checked easily that none of the balanced functions involves an entangling operation—all the operations can be written as products of σ_z and identity matrices. (In the same work, Collins *et al.* give an argument why in the one- and two-qubit case we do not necessarily get a speed-up compared to the classical case—the speed-up depends on the choice of the representation of f.) They conclude that only up from three qubits the implementation of the Deutsch–Jozsa algorithm can be regarded as a demonstration of a system's ability to handle entangling operations, which is essential for quantum computation, and therefore as a demonstration of the feasibility of quantum computation.

In the remaining part of this thesis, we will focus on Collins' version of the Deutsch–Jozsa algorithm without explicitly mentioning it every time.

On the practical meaning of oracles

At the end of the section about the Deutsch algorithm, we would like to counter the prejudice that this algorithm is only of theoretical use or, even worse, is of no use at all, since *we* have to *construct* the oracle and therefore already *know* whether the function is constant or balanced.

Imagine we have some hard decision problem (i.e., a yes/no question) to solve. For instance, we could have a travelling salesman problem (see, e.g., [Mer02]) and the question is: 'Is there a solution of the travelling salesman problem with length < 5?' Solving travelling salesman problems is usually very hard. Now imagine we want to compare two slightly different instances of the travelling salesman problem where the difference is, say, the position of one of the cities. But the only thing we want to find out is whether there is some *difference* between the two instances, i.e., whether one of the two instances performs better than the other in the sense that only one of both has a solution with length < 5.

In order to find out about this, we write some algorithm (i.e., some logic circuit) which takes one Boolean value (i.e., the switch between the two instances) as an input and gives the solution for the decision problem 'length < 5' as a Boolean output. We can therefore restate our problem: we have to find out whether the one-bit Boolean function implemented by our circuit (note that we know the circuit *but not the function*) is constant or balanced. Now assume that the time needed for evaluating the circuit is very long (say, 24 hours[8]). The time will not depend on the input switch since the sequence of operations is fixed. In order to solve our task—finding out whether there is a difference between the two instances of the problem—we would have to evaluate our circuit twice, once for each input value of the switch.

[8]To state things like this is pointless, of course. Its only purpose is to make the reader feel more comfortable. In fact, not the absolute execution time but the scaling behaviour is the big advantage of quantum algorithms.

With Deutsch, however, this circuit only has to be evaluated once! In order to understand this, it is important to know that any normal logic circuit can be transformed into a reversible logic circuit without essentially changing the compexity [Ben73], so that we can devise a reversible logic circuit doing the same task as the classical one (except that it now acts on two lines as the oracle in Fig. 2.2). This reversible logic circuit can also be built on quantum bits without changing its effect on the computational basis states. Now, we can simply find out whether the oracle is constant or balanced (i.e., whether there is some difference between the two instances or not) by only evaluating the oracle *once*. We saved a factor of two!

As already mentioned in a footnote before, this is not the full truth, since the main point is not the execution time in a certain case but the scaling behaviour. Nevertheless, we hope we could emphasize that using oracles in quantum algorithms is *not* a purely theoretical construction.

In the next section, we present Shor's algorithm which can be described in terms of an oracle as well, and where in fact an oracle is known which makes this algorithm to be a very useful one.

2.2.2 Quantum algorithms using the Fourier transform

The Deutsch–Jozsa algorithm belongs to the large class of quantum algorithms based on the Quantum Fourier Transform, which includes the Bernstein–Vazirani algorithm [BV93], Simon's period-finding algorithm [Sim94], and Shor's celebrated algorithm for factoring large numbers in polynomial time [Sho94]. Kitaev [Kit95] gave an alternative description of Shor's algorithm as a solution to the "Abelian stabilizer problem". Subsequently, it has been shown that all these algorithms share the same group theoretical background [Joz98, Høy99]; in fact, they can all be derived from a general algorithm solving the "hidden subgroup problem". In this section, we will only be able to provide a brief introduction to this field of algorithms.

The Quantum Fourier Transform

For a sequence $x_0, \ldots, x_{m-1}$ of m complex numbers, the discrete Fourier transform $y_0, \ldots, y_{m-1}$ of $x_0, \ldots, x_{m-1}$ is defined as

$$y_k = \frac{1}{\sqrt{m}} \sum_{l=0}^{m-1} e^{2\pi i k l/m} x_l \ . \tag{2.8}$$

In analogy to (2.8), we define the Quantum Fourier Transform (QFT) on N qubits ($2^N \equiv m$) as the mapping ($k, l \in \{0, \ldots, 2^N - 1\}$)

$$|l\rangle \stackrel{\mathrm{QFT}}{\longmapsto} \frac{1}{\sqrt{2^N}} \sum_{k=0}^{2^N-1} e^{2\pi i k l/2^N} |k\rangle \ . \tag{2.9}$$

This transformation is in fact unitary, and it can be realized by $O(N^2)$ one- and two-bit operations (see, e.g., [CEMM98]). The connection to the discrete Fourier transform is found by applying the QFT to a superposition of states:

$$\begin{aligned}\sum_{l=0}^{2^N-1} x_l|l\rangle &= \sum_{l=0}^{2^N-1} x_l \left[\frac{1}{\sqrt{2^N}} \sum_{k=0}^{2^N-1} e^{2\pi ikl/2^N}|k\rangle\right] \\ &= \sum_{k=0}^{2^N-1} \left[\frac{1}{\sqrt{2^N}} \sum_{l=0}^{2^N-1} e^{2\pi ikl/2^N} x_l\right] |k\rangle = \sum_{k=0}^{2^N-1} y_k|k\rangle \ .\end{aligned}$$

We thus find that the coefficients $(y_k)_{2^N}$ of the basis states *after* the QFT are exactly the Fourier transformed of the coefficients $(x_l)_{2^N}$ *before* the QFT. The QFT simply accomplished a discrete Fourier transform of the amplitudes of the states.

As mentioned before, the QFT is also a main ingredient of the Deutsch–Jozsa algorithm. To see why, consider the simplest case $N = 1$ of the QFT, i.e., a space with only two elements. In this case, the QFT becomes

$$\mathrm{QFT}_1 : |l\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{k=0}^{1} e^{2\pi ikl/2}|m\rangle$$

which means that it maps

$$\begin{aligned}|0\rangle &\overset{\mathrm{QFT}_1}{\longmapsto} \frac{1}{\sqrt{2}}\Big[|0\rangle + |1\rangle\Big] \text{ and} \\ |1\rangle &\overset{\mathrm{QFT}_1}{\longmapsto} \frac{1}{\sqrt{2}}\Big[|0\rangle - |1\rangle\Big] \ ,\end{aligned}$$

i.e., it is the Hadamard transform.

We thus see that the Hadamard transform is indeed the simplest case of the Quantum Fourier Transform! In fact, the N-qubit Deutsch–Jozsa algorithm performs an N-dimensional Fourier transform with two elements per dimension. If we have a constant function, the $\mathbf{0}$ component of the Fourier tranform is the only component which is not zero, in case we have a balanced function we only have contributions of Fourier components different from $\mathbf{0}$. This is because the contribution of all Fourier components except $\mathbf{0}$ to the mean value is zero, and only the $\mathbf{0}$ component is responsible for a constant offset of the function. But the balanced functions are in fact exactly the functions with mean value zero, and the constant ones those with only $\mathbf{0}$ Fourier component.

Shor's factoring algorithm

The Fourier transform can not only be used for finding the mean value of certain functions. A much more common application of the Fourier transform is to detect

things like the periodicity of a function by analyzing the Fourier spectrum. In fact, this in just what Shor's factoring algorithm utilizes in order to quickly factorize numbers.

Suppose we want to factor a large number $N \in \mathbb{N}$. Classically, there is no algorithm known which can do this in polynomial time (not even probabilistically). Although this does not exclude the possibility that such an algorithm exists, none has been found despite the hard work done on this throughout the past.

But in 1994, Shor [Sho94] could show that there exists a probabilistic quantum algorithm which *can* accomplish this in *polynomial* time.

Before describing the algorithm, a few restrictions have to be pointed out. The algorithm only can factor odd numbers and fails for prime powers, i.e., for $N = p^\alpha$ with p prime. Nevertheless, this is not a serious drawback, since even numbers can be divided by two until the remaining part is odd, and the case $N = p^\alpha$ can also be excluded in polynomial time by taking the corresponding roots. Therefore, we will assume that the number we want to factor is odd and has at least two different prime factors.

In the following, we will give a brief survey of how Shor's algorithm works. Details on this, as the number theoretical background or the estimation of the success probabilities, can be found in Shor's original work [Sho94] or in the review article by Ekert and Jozsa [EJ96].

Let N be the number to be factored. Now choose randomly a number a coprime to N. Define the function

$$\begin{aligned} f : \mathbb{N} &\to \mathbb{N} \\ x &\mapsto f(x) = a^x \bmod N \,, \end{aligned} \tag{2.10}$$

and find the period r of f, i.e., the smallest $r > 0$ with $f(r) = 1$. Then, there is a probability of at least $1/2$ that r is even *and* that $a^{r/2} \pm 1$ is no longer coprime to N, so that we can find a nontrivial factor of N by calculating the greatest common divisor $\gcd(N, a^{r/2} \pm 1)$ of N and $a^{r/2} \pm 1$ (this can be done efficiently using Euclid's algorithm [Euc]).

It can be shown that by recursive application of this method with a randomly chosen a the probability of finding a factor (and, in fact, also of finding *all* the factors) of N converges quickly enough in order to obtain a probabilistic method to factor N with polynomial complexity.

This was known in principle before Shor presented his factoring algorithm. The new thing he did was to devise how the period of f could be determined polynomially, whereas classically (similar to Deutsch's problem!) most of the function values of f have to be calculated in order to find the period.

To this end, choose m such that $N < 2^m$. Define the mapping O_f on two m-bit quantum registers $|x\rangle$ and $|y\rangle$ as

$$|x\rangle|y\rangle \stackrel{O_f}{\longmapsto} |x\rangle|\,(y \cdot (a^x)) \bmod N\rangle \,.$$

This mapping O_f can be implemented efficiently, e.g., by repeatedly squaring a modulo N in order to get a^{2^i} and multiplying this with y (mod N) iff the ith bit of x is set. Recall that calculations like the ones above which can be done efficiently classically can also be carried out efficiently on quantum computers.

Now start by initializing the two m-qubit registers to $|0\rangle$ and $|1\rangle$, respectively. By Hadamard-transforming the first register, one gets

$$|\psi_0\rangle = \sum_{k=0}^{2^m-1} |k\rangle|1\rangle \ .$$

The application of O_f (playing the role of the oracle) transforms this to

$$|\psi_1\rangle = \sum_{k=0}^{2^m-1} |k\rangle|a^k \bmod N\rangle \ .$$

Now measure the second register. Thereby, the state of the second register collapses to the measured value $|y\rangle$ (which lies in the range of f), while the first register collapses to a superposition of all states $|x\rangle$ with $a^x \bmod N \equiv f(x) = y$. In fact, the first register only contains states with a distance equal to the periodicity of f.[9]

Now, the task which remains to be done is to determine the period of the marked states in the first register, i.e., the period r of f. To this end, it is of no use to measure the first register, since this would collapse the state to one fixed value x with $f(x) = y$, giving no information about the periodicity of f. This means we have to find a more sophisticated method to determine the period.

Here, the Quantum Fourier Transform comes into play. By applying the QFT to the first register, we get[10] a state which consists of a superposition of all states $|x\rangle$ with $x = \lambda \cdot 2^m/r$, where $\lambda = 0, \ldots, r-1$ is distributed equiprobably.[11] Thus, the QFT transforms the r-periodicity of the marked states to a $2^m/r$-periodicity and—and *this* is the crucial point—it removes the offset of the original superposition.

If we now measure, we know some λ/r (the value we found in the register). Since λ is chosen equiprobably, we then have a sufficient probability that $\gcd(\lambda, r) = 1$, in which case we can easily determine r.

We thus see that, with sufficiently high probability, we can find the period r of f, which in course allows us to find a nontrivial factor of N with a probability also high enough. For the proofs that the probability is indeed high enough so that the algorithm stays polynomial, we once again refer to [Sho94] and [EJ96].

[9] This is not fully true, though. Since 2^m is not necessarily a multiple of the period, we will not have periodic behaviour at the boundaries. Nevertheless, it can be proven that, if m is chosen large enough, this only gives a negligible error, and therefore will not influence the result too much—recall that our algorithm only succeeds in a probabilistic manner.

[10] we omit the calculation

[11] Once more, this only works approximately if the function is not exactly periodic. See Footnote 9.

2.2.3 Grover's database search algorithm

A very different type of algorithm is Grover's algorithm for searching an unsorted database [Gro96]. Here, the task consists in finding one of serveral marked items in a database, i.e., given a Boolean function f on N qubits, we want to find some $x \in \{0,1\}^N$ such that $f(x) = 1$. Classically, this is hard in case no information about the structure of f is provided, since in that case one basically has to search the database item by item. On the other hand, a lot of interesting hard computational problems can be expressed this way. We only mention the NP-complete problem n-SAT where the task is to determine whether a set of logical OR combinations of n different x_is can be true simultaneously [Mer02].

As a matter of fact, Grover's algorithm only gives a polynomial speed-up: whereas classically $O(N)$ steps are needed, Grover will succeed in $O(\sqrt{N})$ steps.

In Grover's algorithm, the oracle of our database consists of a unitary matrix flipping the phases of all "marked" states, i.e.,

$$|x\rangle \stackrel{U_f}{\longmapsto} (-1)^{f(x)}|x\rangle \ .$$

This type of oracle can be obtained from the classical oracle by the method described in the section about the Deutsch–Jozsa algorithm. In contrast to the Deutsch–Jozsa algorithm, the function f can be an arbitrary Boolean function defined on N bits.

The key operation in Grover's algorithm is the operation $-HG_0H$. Here, H is the Hadamard transform on the N qubits and G_0 is Grover's oracle for $f(x) = \delta_{x,0}$, i.e., it flips exactly the phase of the $|0\rangle$ state. (The minus sign has only been introduced for convenience and is quantum mechanically irrelevant, as it corresponds to a global phase.) Since G_0 can be expressed as

$$G_0 = I - 2|0\rangle\langle 0|$$

and $H^2 = I$, one can simplify

$$\begin{aligned} -HG_0H &= -H\Big(I - 2|0\rangle\langle 0|\Big)H \\ &= 2H|0\rangle\langle 0|H - I \ . \end{aligned}$$

As we will see, only real coefficients to the basis states will appear in Grover's algorithm. Therefore, it is sufficient to analyze what happens to a state

$$|\psi\rangle = \sum_y \alpha_y |y\rangle \ ,$$

where $\alpha_y \in \mathbb{R}$, if we apply $-HG_0H$.

The interesting component of $-HG_0H$ is $2H|0\rangle\langle 0|H$. Applying it to $|\psi\rangle$ yields:

$$\begin{aligned}
2H|0\rangle\langle 0|H|\psi\rangle &= 2H|0\rangle\langle 0|H\textstyle\sum_x \alpha_x|x\rangle \\
&= 2H|0\rangle\langle 0|\textstyle\sum_x \alpha_x \frac{1}{\sqrt{2^N}} \sum_y (-1)^{x\cdot y}|y\rangle \\
&= 2H|0\rangle \textstyle\sum_x \alpha_x \frac{1}{\sqrt{2^N}} \underbrace{\sum_y (-1)^{x\cdot y}\langle 0|y\rangle}_{=(-1)^{x\cdot 0}=1} \\
&= 2H|0\rangle \textstyle\sum_x \alpha_x \frac{1}{\sqrt{2^N}} \\
&= 2\tfrac{1}{2^N} \textstyle\sum_y (-1)^{0\cdot y}|y\rangle \sum_x \alpha_x \\
&= 2\textstyle\sum_y |y\rangle \frac{1}{2^N} \sum_x \alpha_x \ .
\end{aligned}$$

Observe that $\frac{1}{2^N}\sum_x \alpha_x = \langle \alpha_x\rangle_x$, the mean value of all α_x. It follows:

$$\begin{aligned}
-HG_0H|\psi\rangle &= \Big(2H|0\rangle\langle 0|H - I\Big)\textstyle\sum_y \alpha_y|y\rangle \\
&= 2\textstyle\sum_y \langle \alpha_x\rangle_x|y\rangle - \sum_y \alpha_y|y\rangle \\
&= \displaystyle\sum_y \Big[2\langle \alpha_x\rangle_x - \alpha_y\Big]|y\rangle \ .
\end{aligned}$$

Thus, $-HG_0H$ maps the (real!) amplitude α_y of each state $|y\rangle$ to $2\langle \alpha_x\rangle_x - \alpha_y$.

Actually, this mapping has a visual interpretation—it simply *mirrors* the amplitudes α_y about the mean value $\langle \alpha_x\rangle_x$ of *all* amplitudes.[12] With this information, we already can understand how Grover's database search algorithm works. Imagine we want to search a database with only one item marked, i.e., U_f simply flips the sign of only one state $|y_0\rangle$. Now initialize the N-bit register to an equal superposition of all possible states, e.g., by first resetting it to $|0\rangle$ and then applying the Hadamard transform.

We thus start with a state where all α_y are equal. First apply the oracle U_f, which flips the phase of the marked state, i.e., $\alpha_{y_0} \mapsto -\alpha_{y_0}$. Then, by application of $-HG_0H$, all the amplitudes are mirrored about the mean value. This results in an increase of $|\alpha_{y_0}|^2$, whereas all other probability amplitudes are decreased. After some iterations of U_f and $-HG_0H$, we will finally have a state where a measurement will most certainly yield the marked state.

The whole sequence for 4 qubits is illustrated in Fig. 2.3. Each bar belongs to one α_y. The solid line is the zero line, and the dashed line marks the mean value $\langle \alpha_x\rangle_x$. We start with an equibalanced superposition and apply U_f and $-HG_0H$ in turn. In our case, U_f flips the sign of α_5. Since the $-HG_0H$ step flips the amplitudes about their mean value, the mean value is not changed.

[12] The amplitude before had a distance of $\alpha_y - \langle \alpha_x\rangle_x$ from the mean value, and the new amplitude $2\langle \alpha_x\rangle_x - \alpha_y$ has a distance $(2\langle \alpha_x\rangle_x - \alpha_y) - \langle \alpha_x\rangle_x = -(\alpha_y - \langle \alpha_x\rangle_x)$ from the mean value.

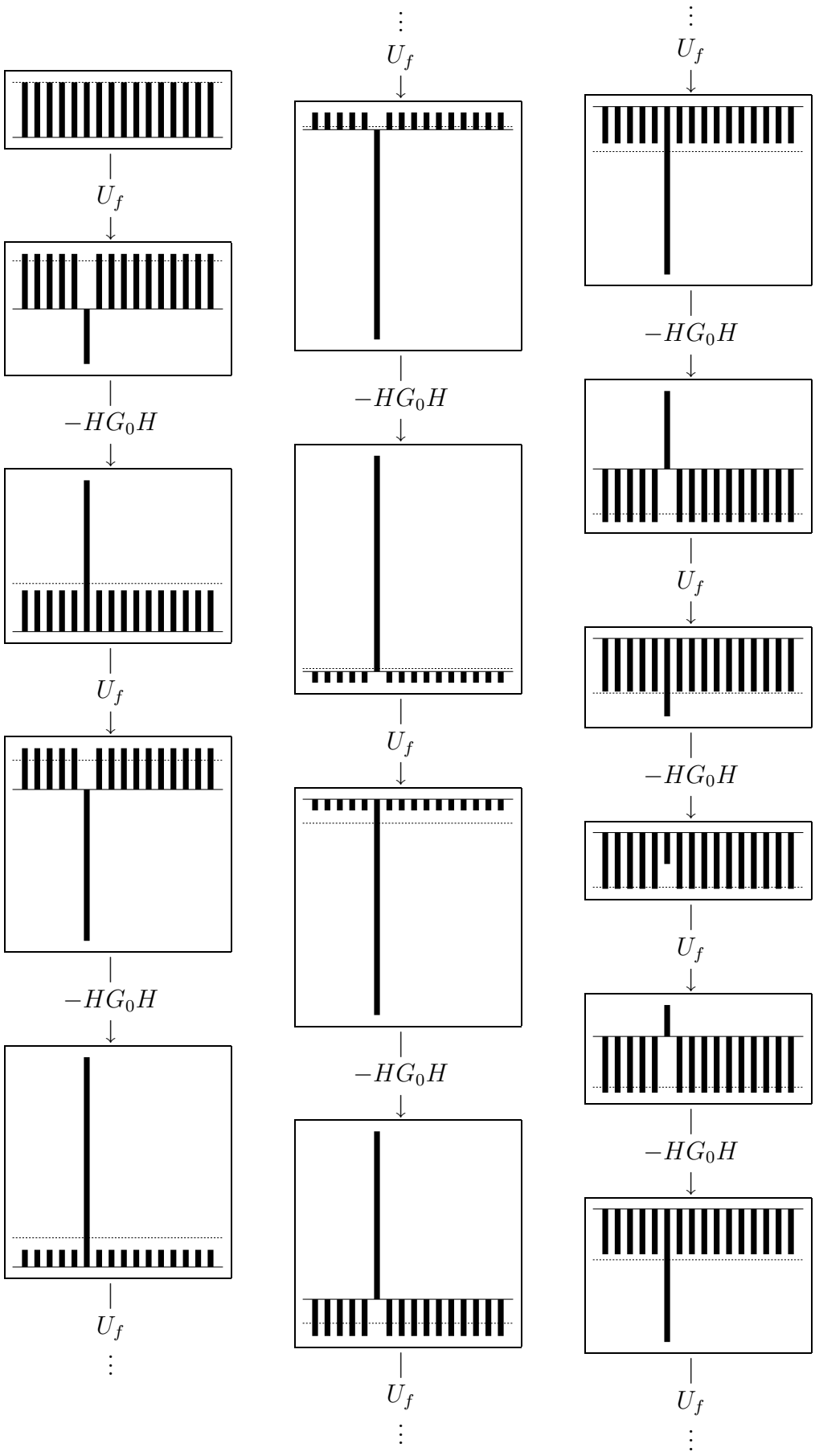


Figure 2.3: Visualization of Grover's algorithm (see text).

At the beginning of the sequence, the amplitude α_5 of the marked state indeed does increase, while the other amplitudes decrease (respect that the square sum of the amplitudes stays constant). After some iterations, though, an optimal ratio is reached—this is where a measurement yields the right outcome with highest probability—and then, the ratio gets worse. There is even some point at which the amplitude of the unmarked states is higher than the amplitude of the marked ones. Finally, the system starts all over—not exactly, though, as can be seen from the figure as well.

We now give the analytic expressions for the α_y, i.e., for the state of the system in the course of Grover's algorithm, in order to determine the number of iterations after which the optimal ratio can be achieved. We only give the results, though; the correctness of these results can be checked straightforwardly.

The state $|\psi_k\rangle$ after the kth application of the oracle and the flipping operation, $-HG_0H \cdot U_f$, where the oracle marks the state x_0, is

$$|\psi_k\rangle = \sum_{x \neq x_0} \frac{1}{\sqrt{N-1}} \cos((2k+1)\theta)|x\rangle + \sin((2k+1)\theta)|x_0\rangle \;, \tag{2.11}$$

where θ is chosen such that $\sin^2\theta = 1/N$. From this formula, one can easily see that in fact the ratio of the amplitudes of marked and unmarked states is oscillating, and there is even some domain where the unmarked states are over-represented.

In order to find the number of iterations when the ratio gets best, we assume that N is large (since we are interested in complexity and thus in the asymptotic behaviour). Then, $\theta \approx \sin\theta = 1/\sqrt{N}$, and the optimal ratio is reached for $\cos(2k+1)\theta = 0$, i.e., the first time for $(2k_0+1)\theta \approx 2k_0\theta = \pi/2$. We thus find that the number of iterations has to be

$$k_0 \approx \frac{\pi}{4}\sqrt{N}$$

or the next integer, so that the number of iterations and thus oracle evaluations goes as $O(\sqrt{N})$, opposed to $O(N)$ for the classical case.

It is clear in principle that this also works for finding one of a couple of marked items. In this case, the number of iterations even gets lower! Nevertheless, in all of these cases it is necessary to *know* the number of marked items in order to measure at the right point. But there exists a method for first estimating the number of marked items as well, thus allowing us to find marked items in $O(\sqrt{N})$ even if we do not know their number. The details on this, as well as a thourough overview over Grover's algorithm and the proof why the speed-up of $O(\sqrt{N})$ vs. $O(N)$ is optimal for a very large class of search algorithms, are dicussed in [BBHT98].

2.2.4 Hamiltonian simulation

A quite different—and somehow the most intuitive—application for quantum computers is the simulation of quantum mechanical systems. In fact, Feynman already pointed out in 1982 [Fey82] that quantum systems could simulate quantum systems in linear time, whereas classical computers need exponential time (actually, this was the first time it was noted that quantum computers could do exponentially better than classical ones).

Although this seems clear and easy to understand, there are a few fundamental problems: first of all, we have to emulate some unitary time evolution on a system of qubits, thus rising the question whether the needed unitary can be built in polynomial time. Secondly, since there is no possibility to read out the full quantum state of the system after the simulation (which *could* be done on a classical computer), we can only find out about certain properties of the system. Particularly, we have to devise ways how to "distill" these properties so that we can measure them.

Despite of these drawbacks, there exist some suggestions for algorithms calculating properties of a physical system. As an example, we briefly present an algorithm [AL99] for finding eigenvalues and eigenvectors of unitaries in polynomial time (as a function of the number of qubits), thus providing an exponential speed-up.

The problem can be stated as follows: given a time evolution $U = e^{-iHt/\hbar}$, find eigenvalues and corresponding eigenvectors of U.

For this, the algorithm requires some initial vector $|\text{in}\rangle$ satisfying that $|\langle \text{in}|v\rangle|^2$ is not exponentially small for all eigenvectors $|v\rangle$ one wants to find. Then, we can find some corresponding eigenvalue λ_v in a time proportional to $1/|\langle \text{in}|v\rangle|^2$ and to the inverse of the accuracy. Additionally, we will also obtain an eigenvector $|v\rangle$ of λ_v with the same accuracy.

In order to see how this works, it is important to note that it is possible to implement time-evolutions of local Hamiltonians in polynomial time on a quantum computer. This does *not* imply that we also know how to generate U^j in a time polynomial in $\log j$.

We start with a total of $m + l$ qubits, where the first register of m qubits will be used for a QFT and the second register is the Hilbert space in which U acts. Define $M = 2^m$. We start with the state

$$\begin{aligned} |\psi_{\text{initial}}\rangle &= H^{\otimes m}|0\rangle|\text{in}\rangle \\ &= \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |j\rangle|\text{in}\rangle \ , \end{aligned}$$

where $|\text{in}\rangle$ is the initial state mentioned above; we can find it by guessing or by using classical approximation methods.

We now apply U^j to the second register, where j is the value in the first register. This can be done by controlled-U^{2^k} operations controlled by the k-th bit of j, and thus needs a time proportional to M. We get the state

$$|\psi_U\rangle = \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |j\rangle U^j |\text{in}\rangle \ . \tag{2.12}$$

If we now rewrite the guessed state $|\text{in}\rangle$ with respect to the eigenstates $|\phi_k\rangle$ (with corresonding eigenvalues λ_k) of U, i.e.,

$$|\text{in}\rangle = \sum_k c_k |\phi_k\rangle \ ,$$

then (2.12) can be rewritten as

$$|\psi_U\rangle = \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |j\rangle U^j \sum_k c_k |\phi_k\rangle \tag{2.13}$$

$$= \frac{1}{\sqrt{M}} \sum_k c_k \sum_{j=0}^{M-1} |j\rangle \lambda_k^j |\phi_k\rangle \ . \tag{2.14}$$

Since U is unitary, the eigenvalues can be rewritten as $\lambda_k = e^{i\omega_k j}$, and we find

$$|\psi_U\rangle = \frac{1}{\sqrt{M}} \sum_k c_k \left[\sum_{j=0}^{M-1} e^{i\omega_k j} |j\rangle \right] |\phi_k\rangle \ . \tag{2.15}$$

By applying a QFT to the first register, each superposition $\sum_{j=0}^{M-1} e^{i\omega_k j}|j\rangle$ is transformed into a state $|\bar{\omega}_k\rangle$, where $\bar{\omega}_k$ is ω_k rounded to m bits. Thus, after the application of the QFT, a measurement will yield the logarithm of the eigenvalue ω_k (with accuracy $\epsilon = 1/M$) in the first register and a corresponding eigenvector in the second register. (The probability for a certain eigenvalue λ_k is $|c_k|^2 = |\langle \text{in}|\phi_k\rangle|^2$ —that's why we require this quantity not to be exponentially small.)

So finally we get an eigenvalue and eigenvector of $U \in \mathrm{U}(2^l)$ with accuracy $\epsilon = 1/M$. The space required goes as $m + l$, which is exponentially better than with a classical algorithm. Since for a large class of applications it is possible to find an implementation of U in $O(\text{poly}(l))$ steps, the execution time scales with the logarithm of the matrix size which gives an exponential advantage in time as well. Moreover, the time required scales as $M = 1/\epsilon$. In case we want to have an exponential advantage with respect to the accuracy, too, we have to find a way how to implement all U^{2^j} in a time polynomial in j.

Chapter 3

Josephson devices for quantum computing

In this chapter we show how superconducting devices can be used as quantum bits. Firstly, we introduce the hardware setup for a single qubit, derive the Hamiltonian, and show why it can be regarded as a two-level system. Then, we present different couplings and discuss their andvantages and disadvantages. Finally, we report on some experiments with Josephson qubits. The chapter closes with a short summary, resuming the results needed for the rest of the work.

As general references for this chapter we mention the book by Grabert and Devoret [GD92] and the review article by Makhlin, Schön and Shnirman [MSS01].

3.1 Josephson charge qubits

Consider the hardware setup in Fig. 3.1. A superconducting island is coupled to a superconducting reservoir via a Josephson junction. Furthermore, the island is coupled capacitively to a voltage source V. The Josephson junction has the Josephson energy E_J and the capacitance C_J, the coupling capacitance to V is C_g.

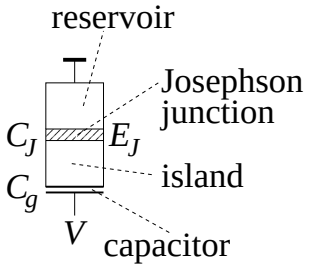


Figure 3.1: Hardware setup under consideration. A superconducting island is coupled by a Josephson junction to a superconducting reservoir and capacitively to a voltage source.

The Josephson junction allows for the tunneling of Cooper pairs. If we work at temperatures low enough to prevent quasiparticle tunneling (i.e., the superconducing energy gap is the largest energy in the system), only Cooper pairs are present on the island. Due to the Josephson tunneling, the net charge on the island can change, i.e., there is some num-

ber n of excess Cooper pairs on the island. The quantum mechanical states with different excess Cooper pair number $|n\rangle$ span a Hilbert space. By properly adjusting the hardware parameters, we can restrict the system to a two-dimensional subspace, thus creating an artificial two-level system, the *Josephson charge qubit*.

Derivation of the Hamiltonian

The Hamiltonian of the system in Fig. 3.1 consists of two parts. There is one contribution from the electrostatic energy of the system, and another one which is due to the Josephson effect.

Firstly, we derive the charging energy. This can be done purely classical, as long as one keeps in mind that charge can be transferred through the capacitor C_J. The transfer mechanism itself is of no importance for the derivation, i.e., we do not have to know that this due to the quantum mechanical tunneling of Cooper pairs.

To this end, consider the classical analogue of the charge qubit given in Fig. 3.2. This system obeys the following equations:

$$V = \frac{Q_g}{C_g} + \frac{Q_J}{C_J} \tag{3.1}$$

$$q = Q_J - Q_g \,, \tag{3.2}$$

where (3.2) defines the excess charge on the island, i.e., the charge which has been transferred through C_J.

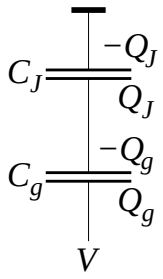


Figure 3.2: Classical analogue of a charge qubit.

The total electrostatic energy of the system is

$$E_{\rm ch} = \frac{Q_J^2}{2C_J} + \frac{Q_g^2}{2C_g} \,. \tag{3.3}$$

Yet, $E_{\rm ch} \equiv E_{\rm ch}(Q_J, Q_g)$ whereas in fact the variables are the transferred charge q and the voltage V. In fact, Q_g comes entirely from the voltage source and therefore is not a variable at all, while following (3.2),

$$Q_J = Q_g + q \,, \tag{3.4}$$

so that Q_J is a sum of Q_g and the variable q. (Note that this breaks the symmetry between Q_g and Q_J. This is only possible since we know that the charge q has been transferred through C_J and *not* through C_g.)

Things get more clear by studying the differential of $E_{\rm ch}$:

$$\mathrm{d}E_{\rm ch} = \frac{Q_g}{C_g}\mathrm{d}Q_g + \frac{Q_J}{C_J}\mathrm{d}Q_J \,, \tag{3.5}$$

and since by (3.4) $\mathrm{d}Q_J = \mathrm{d}Q_g + \mathrm{d}q$, one gets:

$$\begin{aligned} \mathrm{d}E_{\mathrm{ch}} &= \left[\frac{Q_g}{C_g} + \frac{Q_J}{C_J}\right] \mathrm{d}Q_g + \frac{Q_J}{C_J}\mathrm{d}q \\ &\overset{(3.1)}{=} V\mathrm{d}Q_g + \frac{Q_J}{C_J}\mathrm{d}q \; . \end{aligned} \tag{3.6}$$

This result can be understood easily. The first part is the change of the electrostatic energy which is due to the voltage source, and the second part describes the energy change if some charge $\mathrm{d}q$ is transferred through C_J, since $\frac{Q_J}{C_J}$ is the voltage of C_J.

As already mentioned, we would like to have the *free* energy $F_{\mathrm{ch}}(q, V)$, while (3.6) shows us that $E_{\mathrm{ch}} \equiv E_{\mathrm{ch}}(q, Q_g)$. But (3.6) also tells us that

$$F_{\mathrm{ch}}(q, V) = E_{\mathrm{ch}}(q, Q_g) - VQ_g \tag{3.7}$$

is just the appropriate Legendre transform for our purpose.

By inserting the definition (3.3) of E_{ch}, and by eliminating Q_g and Q_J using (3.1) and (3.2), we finally find

$$F_{\mathrm{ch}}(q, V) = \frac{q^2 + 2C_g qV - C_g C_J V^2}{2(C_g + C_J)} \; . \tag{3.8}$$

We want to investigate the quantum dynamics in the Hilbert space spanned by the different charge states $|q\rangle$ of the island. This means that all parts of the Hamiltonian (and hence also of the charging energy) which do not depend on q only contribute a global phase to all states $|q\rangle$ of the island and therefore can be neglected. Consequently, we only need to determine F_{ch} up to terms constant in q. Using this invariance, we find

$$F_{\mathrm{ch}}(q, V) = \frac{(q + C_g V)^2}{2(C_g + C_J)} \; . \tag{3.9}$$

In the case of superconductivity, the charge q (which is the excess charge on the island) is $-2ne$, where n is the number of excess Cooper pairs on the island and $e > 0$. Defining the charging energy scale $\mathcal{E}_{\mathrm{ch}}$ and the 'equilibrium number of Cooper pairs' n_0 as

$$\mathcal{E}_{\mathrm{ch}} = \frac{4e^2}{2(C_J + C_g)} \quad \text{and} \tag{3.10}$$

$$n_0 = \frac{VC_g}{2e} \; , \tag{3.11}$$

respectively, we can rewrite the free charging energy as

$$F_{\mathrm{ch}} = \mathcal{E}_{\mathrm{ch}}(n - n_0)^2 \; . \tag{3.12}$$

The corresponding Hamiltonian is diagonal in the charge basis $\{|n\rangle\}$, with eigenvalues $F_{\rm ch}(n)$.

This result means that the charging energy as a function of n_0, i.e., the voltage V, is a parabola for each different Cooper pair number $|n\rangle$ on the island, see Fig. 3.3.

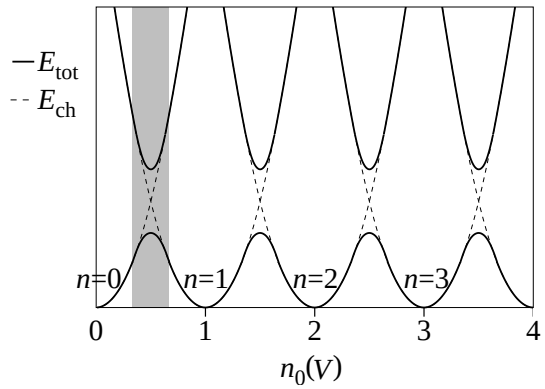


Figure 3.3: Charging energy as a function of n_0, i.e., V, for different excess Cooper pair numbers n (dashed lines); energy eigenstate of the full Hamiltonian (solid lines).

Now we include the Josephson term into the Hamiltonian. For charge qubits, we choose $\mathcal{E}_{\rm ch} \gg E_J \gg k_B T$. Thus, the Josephson effect only adds a small off-diagonal perturbation to the charging part. Therefore, it influences the energy eigenvalues mainly at points of degeneracy, where it mixes the two charge states (see Fig. 3.3).

The tunneling terms can be described as a sum of transitions $|n\rangle \leftrightarrow |n+1\rangle$ with the transition matrix element $-E_J/2$. This leads to the Hamiltonian

$$\mathcal{H}_{\rm full} = \sum_{n=0}^{\infty} \left[\mathcal{E}_{\rm ch}(n - n_0(V))^2 |n\rangle\langle n| - \frac{E_J}{2}(|n\rangle\langle n+1| + |n+1\rangle\langle n|) \right] . \qquad (3.13)$$

Restriction to a two-dimensional subspace

Restrict n_0 to a range near some degeneracy point in Fig. 3.3 (the shaded area). Without loss of generality we assume $V = V_0 + \Delta V$, where $V_0 = e/C_g$ and $\Delta V \ll V$. Then, $n_0 = C_g V/2e \approx 1/2$. If initially the system is in some state $\alpha|0\rangle + \beta|1\rangle$, all states except of $|0\rangle$ and $|1\rangle$ have much higher energies and therefore will not influence the dynamics of the system (note that $\mathcal{E}_{\rm ch} \gg E_J$). This means the system behaves as a two-level system—a qubit. The qubit Hamiltonian is the restriction of (3.13) to the subspace spanned by $\{|0\rangle, |1\rangle\}$:

$$\mathcal{H}' = \mathcal{E}_{\rm ch} \left[\sum_{n=0,1} (n - n_0(V))^2 |n\rangle\langle n| \right] - \frac{E_J}{2} \Big[(|0\rangle\langle 1| + |1\rangle\langle 0|) \Big] . \qquad (3.14)$$

Using the notation introduced in the last paragraph, we see that $n - n_0(V) = n - (1/2 + C_g \Delta V/2e) = (n - 1/2) - C_g \Delta V/2e$. Consequently, $(n - n_0(V))^2$ contains

the quadratic terms $(n-1/2)^2 = 1/4$ ($n \in \{0,1\}$) and $(C_g \Delta V/2e)^2$. Since both expressions do not depend on n, they can be omitted, and the only relevant contribution from $(n - n_0(V))^2$ to the Hamiltonian is $-2(n-1/2)C_g\Delta V/2e$. Using $\sigma_z = |0\rangle\langle 0| - |1\rangle\langle 1|$ and $\sigma_x = |0\rangle\langle 1| + |1\rangle\langle 0|$, the Hamiltonian reads as

$$\mathcal{H} = -\frac{E_z(\Delta V)}{2}\sigma_z - \frac{E_J}{2}\sigma_x \ , \tag{3.15}$$

where $E_z(\Delta V) = -\mathcal{E}_{\rm ch}\, C_g \Delta V/2e$.

We thus have created an artificial quantum mechanical two-level system. Fascinatingly, this is a *macroscopic* quantum mechanical state, since it is generated by the condensate of all superconducting electrons. The usability of this system as a qubit was first suggested by Shnirman, Schön and Hermon [SSH97].

Controllable σ_x part

Our two-level system has a Hamiltonian with a σ_z and a σ_x component, where the σ_z component can be tuned. Although this would be sufficient for applying *any* unitary operation to the qubit, we would like to gain control over the σ_x component, too. Therefore, we have to find a way to make the Josephson energy of the junction tunable. Luckily, there *exists* already a simple[1] way to accomplish this: instead of using a simple Josephson junction, we can replace it by a SQUID loop which is threaded by a flux (Fig. 3.4) [MSS99]. This flux can be used to adjust the Josephson energy properly, giving us full control over our Hamiltonian. Thus, we finally find the following Hamiltonian for our Josephson charge qubit:

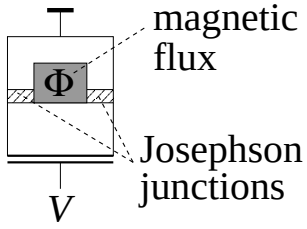


Figure 3.4: Modified Josephson qubit (cf. Fig. 3.1). The Josephson junction has been replaced by a SQUID. By changing the flux Φ, the Josephson energy E_J can be controlled.

$$\mathcal{H} = -\frac{E_z(V)}{2}\sigma_z - \frac{E_x(\Phi)}{2}\sigma_x \ , \tag{3.16}$$

where we replaced E_J by E_x.

Initialization and measurement

Two important requirements for quantum bits, besides sufficient control over the Hamiltonian, are the possibility to initialize the qubit to some well-defined state as well as to measure the qubit in some basis [DiV00]. In the following, we

[1] in a theoretical sense, of course!

mention the method also used by Nakamura *et al.* [NPT99]. Of course, there exist more sophisticated schemes for measurement.

For the initialization, drive the system to $V = 0$, and let it relax into its ground state. By suddenly switching the voltage to the working point, the qubit is in the initial state $|0\rangle$.

For the read-out process, one similarly drives the system to $V = 0$. If the system is in the $|1\rangle$ state, one can detect a Cooper pair by a probe connected to the island. By repeatedly performing the initialization–computation–readout cycle, one measures a current through the probe which corresponds to the probability amplitude of $|1\rangle$.

Finally, we mention that a good measurement method already includes a state preparation. After the measurement, the qubit is in one of two orthogonal states—namely the one which was measured.

3.2 Coupling charge qubits

In order to build powerful quantum computers, it is not enough to have control over the individual qubits. We also need some coupling between the qubits in order to entangle them. In this section, we describe various proposed couplings for Josephson charge qubits.

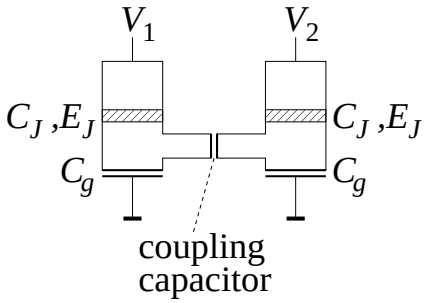


Figure 3.5: Two charge qubits coupled capacitively. The coupling can be extended to a larger number of qubits, although there are natural limitations by the geometry of the system, thus making it hard to couple one qubit to more than a few other qubits. The qubit can be replaced by the SQUID version of Fig. 3.4.

3.2.1 Capacitive coupling

The simplest type of coupling is the capacitive coupling [PYA+02]. To this end, we connect the superconducting islands of two charge qubits by a capacitor (see Fig. 3.5).

The coupling Hamiltonian will clearly be diagonal, since the coupling energy is given by the additional charging energy of the system. We can determine this quantity by a circuit analogous to the one used for the single qubit, see Fig. 3.6. To keep the calculation reasonably simple, we will assume that the two qubits are identical, i.e., they have identical values for the capacitances C_J and C_g as well as for the Josephson energy E_J.

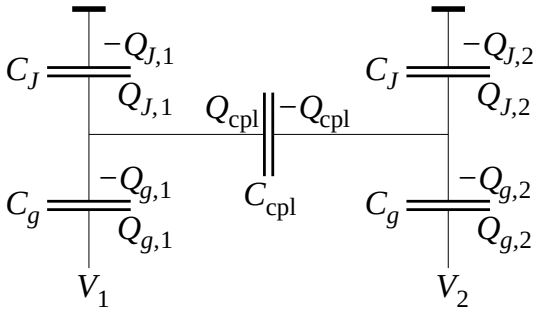


Figure 3.6: Classical circuit corresponding to the capacitively coupled qubits. Cf. also Fig. 3.2.

The system obeys the following equations

$$V_1 = \frac{Q_{g,1}}{C_g} + \frac{Q_{J,1}}{C_J} \tag{3.17}$$

$$V_2 = \frac{Q_{g,2}}{C_g} + \frac{Q_{J,2}}{C_J} \tag{3.18}$$

$$V_1 = \frac{Q_{g,1}}{C_g} + \frac{Q_{\rm cpl}}{C_{\rm cpl}} + \frac{Q_{J,2}}{C_J} \tag{3.19}$$

$$q_1 = Q_{J,1} - Q_{g,1} + Q_{\rm cpl} \tag{3.20}$$

$$q_2 = Q_{J,2} - Q_{g,2} - Q_{\rm cpl} \ . \tag{3.21}$$

Here, q_1 and q_2 are the excess Cooper pair charges on the two islands.

We proceed analogously to the derivation of the one-qubit charging energy. We have

$$E_{\rm ch} = \frac{Q_{J,1}^2}{2C_J} + \frac{Q_{J,2}^2}{2C_J} + \frac{Q_{g,1}^2}{2C_g} + \frac{Q_{g,2}^2}{2C_g} + \frac{Q_{\rm cpl}^2}{2C_{\rm cpl}} \ , \tag{3.22}$$

which leads to the differential

$$\mathrm{d}E_{\rm ch} = V_1 \mathrm{d}Q_{g,1} + V_2 \mathrm{d}Q_{g,2} + \frac{Q_{J,1}}{C_J}\mathrm{d}q_1 + \frac{Q_{J,2}}{C_J}\mathrm{d}q_2 \ . \tag{3.23}$$

(We eliminated the $\mathrm{d}Q_{J,i}$s using (3.20) and (3.21). As a "side effect", the $\mathrm{d}Q_{\rm cpl}$ contribution vanished.) The expression can be understood the same way as the one-qubit differential (3.6): the first two terms arise due to the work done by the voltage sources, the other two give the energy change due to Cooper pair tunneling.

From the differential $\mathrm{d}E_{\rm ch}$, we see that we get the free energy by the Legendre transform

$$F_{\rm ch}(q_1, q_2, V_1, V_2) = E_{\rm ch}(Q_{J,1}, Q_{J,2}, Q_{g,1}, Q_{g,2}) - V_1 Q_{g,1} - V_2 Q_{g,2} \ . \tag{3.24}$$

After a lengthy calculation, we find (up to constants which do not depend on the q_is):

$$F_{\rm ch} = \frac{(q_1 + C_g V_1)^2}{2(C_g + C_J)} + \frac{(q_2 + C_g V_2)^2}{2(C_g + C_J)} + \frac{C_{\rm cpl}\left((q_1 + C_g V_1) - (q_2 + C_g V_2)\right)^2}{2(C_g + C_J)(2C_{\rm cpl} + C_g + C_{\rm J})} \ . \tag{3.25}$$

With $\mathcal{E}_{\rm ch}$ from (3.10) and the definitions (cf. (3.11))

$$\begin{aligned} \mathcal{E}_{\rm cpl} &= \frac{(2e)^2 C_{\rm cpl}}{2(C_g+C_J)(2C_{\rm cpl}+C_g+C_J)} \qquad (3.26)\\ &= \frac{C_{\rm cpl}}{2C_{\rm cpl}+C_g+C_J}\mathcal{E}_{\rm ch} \\ n_{0,i}(V) &= C_g V_i/2e\ , \qquad (3.27) \end{aligned}$$

and the number of excess Cooper pairs on the ith island

$$n_i \equiv q_i/(-2e)\ , \qquad (3.28)$$

this corresponds to

$$\begin{aligned} F_{\rm ch} &= \mathcal{E}_{\rm ch}\Big[n_1 - n_{0,1}(V_1)\Big]^2 + \mathcal{E}_{\rm ch}\Big[n_2 - n_{0,2}(V_2)\Big]^2 + \\ &\quad + \mathcal{E}_{\rm cpl}\Big[(n_1 - n_{0,1}(V_1)) - (n_2 - n_{0,2}(V_2))\Big]^2\ . \end{aligned} \qquad (3.29)$$

The first two parts are the contributions we already found in (3.12) for the single qubits, and the third part is the contribution from the coupling.

As for the single qubits, we assume we are working near $n_{0,i}(V) \approx 1/2$, i.e., we now define ΔV_i by $n_{0,i}(V) = 1/2 + C_g\Delta V_i/2e$. Once more, we rewrite $(n_i - n_{0,i}(V)) = (n_i - 1/2) - C_g\Delta V_i/2e$, and use the fact that the squares of the two terms do not depend on $n \in \{0,1\}$. We find

$$\begin{aligned} F_{\rm ch} &= -2\left[(\mathcal{E}_{\rm ch}+\mathcal{E}_{\rm cpl})\frac{C_g\Delta V_1}{2e} - \mathcal{E}_{\rm cpl}\frac{C_g\Delta V_2}{2e}\right]\left(n_1 - \frac{1}{2}\right) - \\ &\quad -2\left[(\mathcal{E}_{\rm ch}+\mathcal{E}_{\rm cpl})\frac{C_g\Delta V_2}{2e} - \mathcal{E}_{\rm cpl}\frac{C_g\Delta V_1}{2e}\right]\left(n_2 - \frac{1}{2}\right) - \\ &\quad -2\,\mathcal{E}_{\rm cpl}\left(n_1 - \frac{1}{2}\right)\left(n_2 - \frac{1}{2}\right) \end{aligned} \qquad (3.30)$$

The first and the second line are the one-qubit parts of the Hamiltonian (except for the Josephson term). The energies have been shifted compared to the one-qubit case (3.15). Now, for the ith qubit, $E_{z,i}$ depends on *both* voltages, $E_{z,i}(\Delta V_1, \Delta V_2) = -(\mathcal{E}_{\rm ch}+\mathcal{E}_{\rm cpl})C_g\Delta V_i/2e + \mathcal{E}_{\rm cpl}C_g\Delta V_j/2e$ $(j \neq i)$. Nevertheless, if $C_{\rm cpl} \ll (C_g + C_J)$, the one-qubit energy shift is rather small.

Finally, the last part of (3.30) yields an interaction Hamiltonian

$$\mathcal{H}_{\rm int} = -\frac{\mathcal{E}_{\rm cpl}}{2}\begin{pmatrix} 1 & & & \\ & -1 & & \\ & & -1 & \\ & & & 1 \end{pmatrix} = -\frac{\mathcal{E}_{\rm cpl}}{2}\sigma_z \otimes \sigma_z\ . \qquad (3.31)$$

Thus, we have obtained a ZZ coupling between the qubits.

As a matter of fact, the capacitive coupling is not tunable, so that all the qubits are coupled all the time. On the other hand, capacitive coupling of charge qubits is relatively easy to achieve—in fact, it will appear in any setup, since there will always be some capacitance between the qubits.

The problem of fixed interactions can be overcome by techniques used in liquid state NMR, where the coupling is also of the ZZ type and cannot be switched. These techniques are well developed [LCYY00], but they require $E_J \gg \mathcal{E}_{\rm cpl}$. Since the coupling energy must be larger than the thermal noise, we need $\mathcal{E}_{\rm ch} \gg E_J \gg \mathcal{E}_{\rm cpl} \gg k_{\rm B}T$, so that we have to work at very low temperatures, and only have a very weak coupling.

3.2.2 Inductive coupling

As we have seen in the previous section, a tunable coupling is highly desireable. It could be operated much more easily and would allow for fast two-bit operations.

The first scheme for the tunable coupling of Josephson charge qubits was proposed by Makhlin, Schön and Shnirman [MSS99]. They suggested to couple the qubits by a joint harmonic oscillator (see Fig. 3.7). By tuning the Josephson energies $E_J^{(i)}$, the qubits can be brought to resonance with the inductor L, thus yielding a coupling Hamiltonian

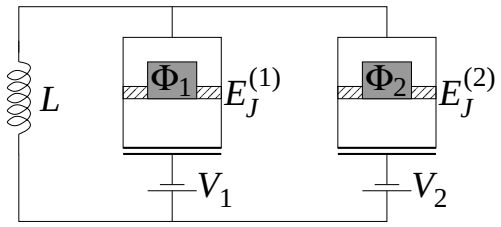


Figure 3.7: Josephson charge qubits with inductive coupling. The coupling is accomplished by a joint harmonic oscillator degree of freedom, which can be switched on by tuning the two Josephson energies correspondingly.

$$\mathcal{H}_{\rm cpl} = -\sum_{i<j} \frac{E_J^{(i)} E_J^{(j)}}{E_L} \sigma_y^{(i)} \sigma_y^{(j)} \ ,$$

where

$$E_L = \frac{(h/2e)^2}{(\pi^2 L)} \left[\frac{C_J}{\left(C_g^{-1} + C_J^{-1}\right)^{-1}} \right] \ .$$

We thus have a tunable $\sigma_y \sigma_y$ coupling, which indeed is equivalent to the ZZ type interaction we found for the capacitive coupling. In contrast to the capacitive coupling, the inductive coupling is tunable; moreover, it allows for easy coupling of distant qubits.

Nevertheless, there are also some disadvantages with this coupling: first of all, since the coupling is switched by the product of the Josephson energies which also determine the σ_x part of the Hamiltonian, we cannot switch on solely one

interaction: we will always have a σ_x one-particle Hamiltonian for the two qubits involved in the coupling.

The second drawback is that this type of interaction does not allow for simultaneous execution of more than one interaction: since the amplitude of the interaction between i and j is determined by $E_J^{(i)} E_J^{(j)}$, it is impossible to activate the interaction for two distinct pairs of qubits alone.

Finally, the coupling constant depends on the total capacitance of the system. In fact, it turns out that it scales as $1/\sqrt{N}$, which makes it impractical for large-scale quantum computers.

3.2.3 Coupling by Josephson junctions

A third method how to couple Josephson charge qubits has been proposed by Siewert *et al.* [SFPS00]. In their proposal, the coupling is accomplished by a SQUID loop, too, thus giving an additional degree of freedom (the flux in the SQUID) which allows for an independent control of each coupling separately (see Fig. 3.8). If we assume that the Josephson energy $E_{J,\mathrm{cpl}}$ of the coupling is much smaller than the charging energy and thus does not influence the two-level property of the qubit, the coupling Hamiltonian can be written as

$$\mathcal{H}_{\mathrm{cpl}} = -\frac{E_{J,\mathrm{cpl}}}{2}\Big[|0\rangle\langle 1| \otimes |1\rangle\langle 0| + |1\rangle\langle 0| \otimes |0\rangle\langle 1|\Big] ,$$

desribing the Josephson tunneling of the excess Cooper pair from one qubit to the other. This can be rewritten in terms of Pauli spin matrices as

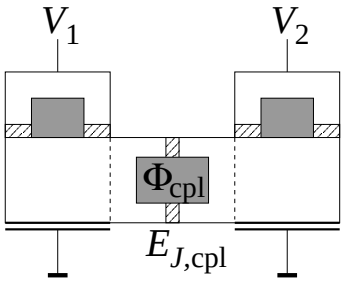


Figure 3.8: Josephson charge qubits coupled by Josephson junctions. The coupling is accomplished by another SQUID loop, thus giving independent control over each coupling separately.

$$\mathcal{H}_{\mathrm{cpl}} = -\frac{E_{J,\mathrm{cpl}}}{4}\left(\sigma_x^{(i)}\sigma_x^{(j)} + \sigma_y^{(i)}\sigma_y^{(j)}\right)$$

for each pair (i, j) of coupled qubits. Compared to the capacitive or inductive coupling, we get an XY interaction Hamiltonian instead of the ZZ interaction we had before.

The coupling by Josephson junctions provides an independently tunable coupling, i.e., we can switch each coupling without any effect on the other couplings or the one-bit Hamiltonian. A drawback of this coupling is the fact that—similar to the capacitive coupling—we have to face geometric restrictions so that we will not be able to provide coupling between arbitrary qubits in most cases. A coupling only between nearest neighbors seems much more likely.

Another problem with the capacitive coupling is the capacitance of the coupling junction: we have to design the Josephson junction such that the capacitance and therefore the capacitive coupling constant is small compared to the Josephson energy of the junction, so that the *capacitive* coupling is only a small perturbation and thus can be neglected to a good approximation.

The energy scales in this case are as follows: $\mathcal{E}_{\text{ch}} \gg (E_J, E_{J,\text{cpl}}) \gg (k_B T, \mathcal{E}_{\text{cpl}})$, where $E_{J,\text{cpl}}$ and $\mathcal{E}_{\text{cpl}}$ are the Josephson and the capacitive energy of the coupling SQUID, respectively. Alternatively, one might want to have $\mathcal{E}_{\text{cpl}} \gg k_B T$, such that the effect of the capacitive coupling can be corrected.

In the remaining part of this thesis, we will mainly focus on the coupling by SQUID loops as a concrete realization. Thereby, we will neglect the capacitance of the coupling junctions and only consider a controllabe XY type interaction Hamiltonian. The geometrical restrictions will result in the assumption that only nearest neighbors can be coupled, thus leading to either a linear or a circular setup of qubits with only nearest neighbor coupling.

3.3 Other Josephson devices as qubits

Of course, Josephson charge qubits are not the only devices which can be used as superconducting qubits. Recall that, for the charge qubits, we assumed that the charging energy was the dominating energy in the system. On the other hand, it is also possible to build a circuit where the Josephson energy dominates. In this case, the variable conjugated to the charge, the phase difference across the Josephson junction—which is proportional to the flux—would be the genuine choice for a basis, and no longer the charge of some island.

This type of qubit is called *flux qubit* [CL83, MOL+99]. The simplest possible version is shown in Fig. 3.9. It consists of a superconducting loop interrupted by a Josephson junction and threaded by some external flux Φ_{ex}. The total Hamiltonian consists of three contributions: one from the Josephson coupling, one magnetic contribution due to the self-inductance of the loop, and finally the charging contribution:

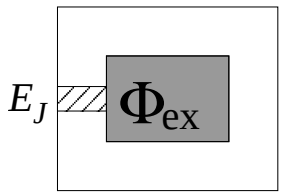


Figure 3.9: The rf-SQUID, the simplest Josephson flux qubit.

$$\mathcal{H} = -E_J \cos\left(2\pi\Phi/\Phi_0\right) + \frac{(\Phi - \Phi_{\text{ex}})^2}{2L} + \frac{Q^2}{2C_J} \,. \tag{3.32}$$

Here, Φ is the flux in the loop, and $2\pi\Phi/\Phi_0 = \varphi$ is the phase difference across the junction. The charge Q is conjugate to Φ, and $\Phi_0 = h/2e$ is the flux quantum. L is the self-inductance of the loop, and C_J is the capacitance of the junction.

Thus, in the flux basis, our Hamiltonian is a Schrödinger operator with a potential energy consisting of the Josephson tunneling term and the inductive

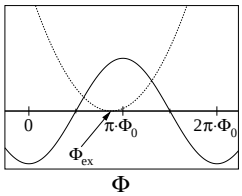


Figure 3.10: Effective potential $V(\Phi)$ in the Hamiltonian (3.32). The solid line shows the Josephson part of the potential, the dashed line the magnetic contribution. Φ_{ex} is the external flux, and Φ_0 is the flux quantum.

term, as shown in Fig. 3.10. Clearly, by choosing L sufficiently large, and $\Phi_{\text{ex}} \approx \Phi_0/2$, this yields a double-well potential. At sufficiently low temperatures, only the two lowest states in the two wells contribute to the physics of the system. We thus get a reduced two-level Hamiltonian with a σ_z part governed by the asymmetry of the well, i.e., $\Phi_{\text{ex}} - \Phi_0/2$, and a σ_x part which depends on the tunneling amplitude between the two wells, i.e., the height E_J of the barrier. Similar to the charge qubit, we can make the σ_x part controllable by replacing the Josephson junction by a SQUID loop.

So in fact, our two-level system is constituted by the flux in the loop which fluctuates around the external flux Φ_{ex}. There are various more sophisticated suggestions for flux qubits [MOL$^+$99], aiming to make them less susceptible to external noise. Also, there exists proposals for qubits residing in the domain *between* charge and flux qubit, i.e., with comparable charging and Josephson contributions to the energy. We will present one of them in the next section.

3.4 Experimental realizations

3.4.1 Nakamura's Josephson charge qubit

The first experimental result demonstrating the feasibility of controlled coherent quantum dynamics with Josephson charge qubits was presented by Nakamura *et al.* [NPT99]. They built a single qubit with the junction implemented as a SQUID loop (Fig. 3.4) and showed that it could perform coherent oscillations (i.e., Rabi oscillations).

In his experiment, Nakamura initially biased the system to a point far from degeneracy, so that it could relax to the $|0\rangle$ state. Then, he applied a gate voltage pulse of length Δt to the gate. The voltage was chosen to drive the system to the point of degeneracy. Thus, during the time Δt the system could evolve coherently with a period depending on the Josephson energy (and thus on the flux). At the end of the pulse, the system was driven back to the initial state, and by the current flowing through a probe junction they could detect whether the qubit was in the $|0\rangle$ or in the $|1\rangle$ state.[2] In order to get a measurable current, an array of pulses was applied, with the time between the pulses much longer than the relaxation

[2]In fact, the probe junction was one of the sources of systematic errors in this experiment, since it was connected to the qubit *all the time*. Single electron transistors (SETs) would

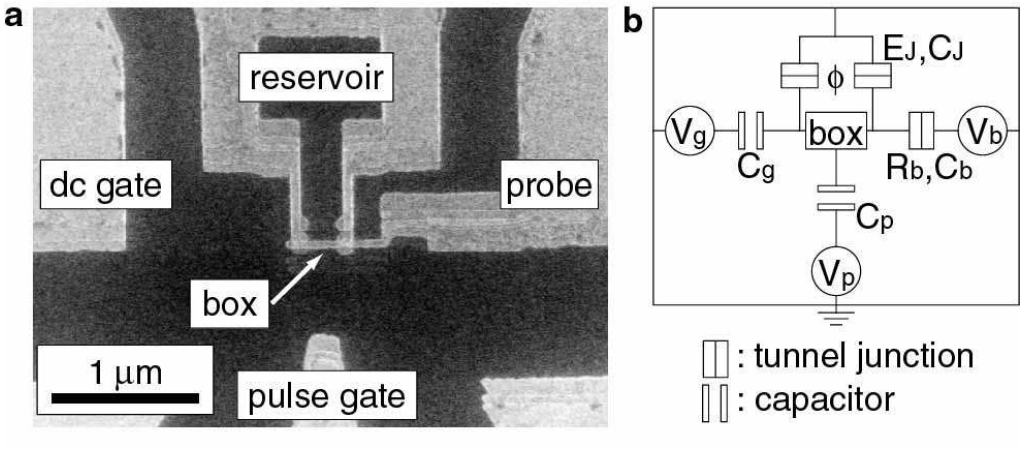


Figure 3.11: Nakamura's experimental setup. (The figure is taken from the e-print version of [NPT99].)

(i.e., measurement) time T_r. The measured current is thus proportional to the probability amplitude of the $|1\rangle$ state. They applied pulses between 80ps and 450ps and could observe coherent oscillations. They also could measure $E_J(\Phi)$ indirectly via the oscillation period, verifying the law $E_J(\Phi) = E_J(0)\cos(\pi\Phi/\Phi_0)$ very well (Φ_0 is the flux quantum). With a flux yielding a period of the Rabi oscillations of about 100ps, they could observe coherent evolution for up to 2ns. This, of course, is by far not sufficient for quantum computation. Nevertheless, there is a lot of space left for improvement, especially concerning the measuring device, as already mentioned before.

The experiment was carried out at $T \approx 30\text{mK} \equiv 3\mu\text{eV}$, the charging energy was $e^2/2C_\Sigma = 117 \pm 3\mu\text{eV}$. Here, C_Σ is the total capacitance of the island. The superconducting gap was $\Delta = 230 \pm 10\mu\text{eV}$, and the Josephson energy E_J about $50\mu\text{eV}$.

Recently, Pashkin *et al.* [PYA+02] presented experimental results demonstrating coherent osciallations in two capacitively coupled charge qubits.

3.4.2 The Saclay qubit

The most promising implementation of a superconducting qubit up to now is the "quantronium" of the Saclay group [VAC+02]. In their experiment, they studied a Josephson qubit with $E_J \sim \mathcal{E}_{\text{ch}}$, i.e., neither n nor the phase φ across the junction are good quantum numbers. Nevertheless, the ground and the first excited state form a two-level system at $n_0 = 1/2$. The quantum state is manipulated by applying microwave pulses to the gate.

The main new point about the Saclay qubit is the readout: the single junc-

provide a possibility to switch the measurement device on and off [MSS01].

tion of the island has been split into two identical junctions in order to form a superconducting loop. The different states of the island are discriminated not by the charge n of the island but by the supercurrent in the loop. This is achieved by entangling the system with the phase of a large Josephson junction ($E_{J,\mathrm{large}} \approx 20 E_{J,\mathrm{small}}$). The readout can be accomplished by applying a pulse to the parallel combination of the junctions, thus changing the current in the large junction, which then can be read out.

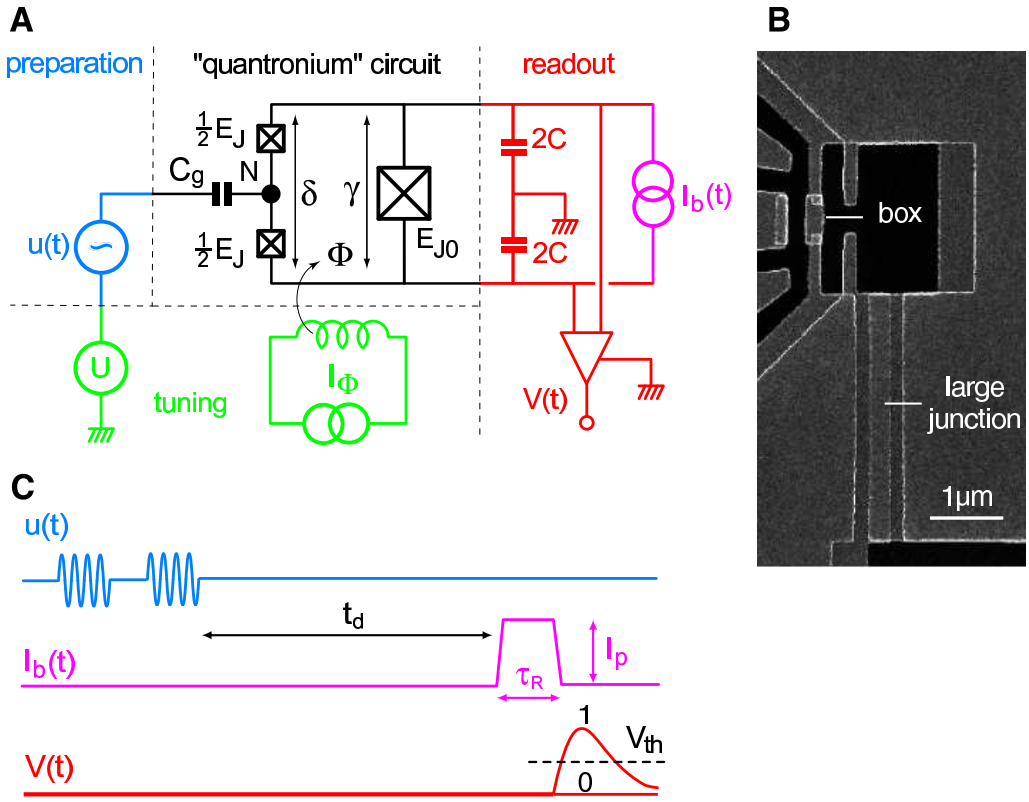


Figure 3.12: The Saclay qubit (The figure is taken from the e-print version of [VAC+02].)

The experiment was carried out at a temperature of about 15mK. $E_J \approx 75\mu$eV, $\mathcal{E}_{\mathrm{ch}} \approx 58\mu$eV. The period of the Rabi oscillations was about 50ns, and the observed decoherence time was 0.50μs. Since the precession period of the system was only about 60ps, this corresponds to about 8000 free coherent oscillations. Still, the time scale for operations manipulating the state of the circuit is of the order of the Rabi frequency, so that this ratio—which is the relevant one—is much lower.

3.5 Summary

We briefly summarize the results of this chapter which are necessary for the rest of this work.

We will investigate qubits with a two-qubit Hamiltonian

$$\mathcal{H} = -\frac{E_z}{2}\sigma_z - \frac{E_x}{2}\sigma_x \,, \tag{3.33}$$

where E_z and E_x can be controlled independently for each qubit.

For the interaction Hamiltonian, we will mainly focus on the XY interaction

$$\mathcal{H} = -\frac{E_{\mathrm{cpl}}}{4}\Big[\sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y\Big] \,. \tag{3.34}$$

The interaction range will be set to the "minimum requirement", i.e., we will only assume couplings between nearest neighbors, where the qubits are arranged in a line or in a ring.

Chapter 4

Universal quantum computation and the XY interaction

In the preceding chapter, we have seen how one can build qubits, i.e., quantum mechanical two-level systems, using Josephson devices. Especially, we derived expressions for the single qubit Hamiltonian as well as for the coupling Hamiltonian between the qubits. It also became apparent that—depending on the physical realization of coupling—there often exist physical restrictions on the pairs of qubits which can be coupled directly.

In Chapter 2, we discussed quantum computation and quantum algorithms. Thereby, the basic building blocks were unitary multi-qubit operations. We noted that there exist so-called universal sets of gates which allow for the construction of arbitrary multi-bit operations. One special set consists of the CNOT operation (between all pairs of qubits) together with arbitrary one-bit operations.

This chapter is intended to link these two chapters. We demonstrate how the universal set mentioned above—CNOT and arbitrary local unitaries—can be generated using the Hamiltonians derived in Chapter 3. Especially, we give hints how to efficiently overcome the natural restrictions concerning the coupling of the qubits.

Main results of this chapter (esp. Sections 4.2, 4.4 and 4.5) have been published in [SS02b].

4.1 One-qubit operations

It this section, we show how arbitrary one-bit operations can be assembled using the one-bit Hamiltonian of the Josephson charge qubit. Recall that the local Hamiltonian (see (3.16)) was of the form

$$\mathcal{H} = -\frac{E_z}{2}\sigma_z - \frac{E_x}{2}\sigma_x \ . \tag{4.1}$$

Here, both E_z and E_x can be controlled independently.

This Hamiltonian directly provides two classes of operations, namely the x rotations,

$$R_x(\phi) = e^{-i\sigma_x\phi/2} = \begin{pmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{pmatrix} ,$$

and the z rotations,

$$R_z(\phi) = e^{-i\sigma_z\phi/2} = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix} .$$

In the following, we show that any local unitary $U \in \mathrm{U}(2)$ can be generated using two z and one x rotation. To this end, we first note that it is sufficient to take $U \in \mathrm{SU}(2)/\{-1, 1\}$, since any unitary matrix is equivalent to some $U \in \mathrm{SU}(2)$ up to a global phase factor (which is physically irrelevant), and for the same reason two matrices in SU(2) which only differ by a factor of -1 are equivalent.

Now it is well known that $\mathrm{SU}(2)/\{-1, 1\}$ is isomorphic to SO(3), the group of rigid-body rotations in three dimensions, by identifying a rotation around the axis $\boldsymbol{\varphi}$ about an angle of $|\boldsymbol{\varphi}|$ with the SU(2) matrix $\exp\left[-i\,\boldsymbol{\varphi}\cdot\boldsymbol{\sigma}/2\right]$.

As one can easily check, the mapping given above exactly identifies R_z and R_x with the rotations about the z and the x axis, while an arbitrary unitary U gives some arbitrary rotation (all of this neglecting global phases). But it is well known, e.g., from classical mechanics, that rotations in three dimensions can be described as a z rotation, an x rotation and another z rotation, where the rotation angles are the so-called *Euler angles*. Thus, we see that x and z rotations—and therefore our one-bit Hamiltonian (4.1), which is capable of generating those rotations—are sufficient to generate any arbitrary one-bit operation.

To state this in more detail, every matrix $U \in \mathrm{U}(2)$ has a (not necessarily unique) decomposition

$$\begin{aligned} U &= e^{-i\varphi} \begin{pmatrix} e^{-i(\alpha+\beta)/2}\cos\theta/2 & e^{-i(\alpha-\beta)/2}\sin\theta/2 \\ -e^{i(\alpha-\beta)/2}\sin\theta/2 & e^{i(\alpha+\beta)/2}\cos\theta/2 \end{pmatrix} \\ &= R_z\left(\alpha - \pi/2\right) R_x\left(\theta\right) R_z\left(\beta + \pi/2\right) . \end{aligned}$$

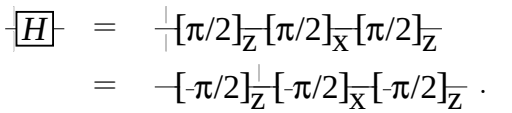
To give an example, the Hadamard transform

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

has the following possible values for α, β and θ:

α	π	0	$-\pi$	0
β	0	$-\pi$	0	π
θ	$-3\pi/2$	$-\pi/2$	$\pi/2$	$3\pi/2$

leading to the following two possible sequences for the Hadamard transform:



4.2 Building the CNOT operation

In this section we illustrate that, depending on the available coupling term $\mathcal{H}_{i,j}$, it is more or less difficult to generate the CNOT operation. We also provide explicit constructions.

We focus on three different types of interaction, two of which we can construct for Josephson charge qubits. The third type is added for reasons of completeness, since it fits well to the two other types.

Firstly, there is the ZZ interaction

$$\mathcal{H}_{i,j}^{ZZ}(E_{i,j}^{ZZ}) = -\frac{E_{i,j}^{ZZ}}{4}\sigma_z^{(i)}\sigma_z^{(j)} \ ,$$

which, beyond by inductively coupled Josephson charge qubits, can also be realized for Josephson flux qubits [OMT+99]. Secondly, the JJ or Heisenberg interaction

$$\mathcal{H}_{i,j}^{JJ}(E_{i,j}^{JJ}) = -\frac{E_{i,j}^{JJ}}{4}\left[\sigma_x^{(i)}\sigma_x^{(j)} + \sigma_y^{(i)}\sigma_y^{(j)} + \sigma_z^{(i)}\sigma_z^{(j)}\right] \ ,$$

which basically appears in systems where spins are coupled by the exchange interaction, for example spins in quantum dots interacting via a tunnel junction [LD98], nuclear spins in phosphorus-doped silicon devices [Kan98], or spin-resonance transistors [VYW+00]; and finally the XY interaction

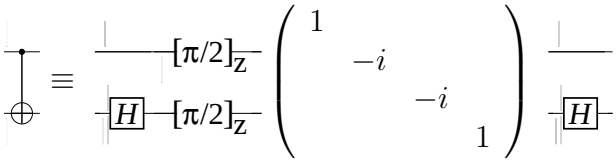
$$\mathcal{H}_{i,j}^{XY}(E_{i,j}^{XY}) = -\frac{E_{i,j}^{XY}}{4}\left[\sigma_x^{(i)}\sigma_x^{(j)} + \sigma_y^{(i)}\sigma_y^{(j)}\right] \ .$$

In addition to Josephson charge qubits coupled by Josephson junctions, this type of coupling has also been proposed for quantum dot spins coupled by a cavity [IAB+99] and for nuclear spins interacting via a two-dimensional electron gas [MPG01].

For the ZZ-interaction, the CNOT operation is indeed the natural two-bit operation since

$$\exp\left[-i\mathcal{H}_{i,j}^{ZZ}(E_{i,j}^{ZZ})\frac{\pi}{E_{i,j}^{ZZ}}\right] = e^{i\pi/4}\begin{pmatrix} 1 & & & \\ & -i & & \\ & & -i & \\ & & & 1 \end{pmatrix}$$

is equivalent to CNOT up to one-bit operations [Mak00], e.g., by



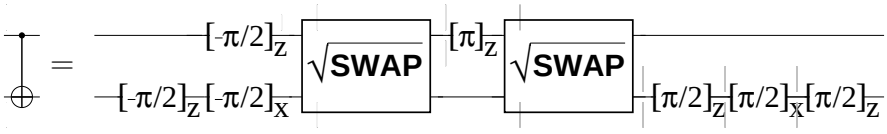
As opposed to this, the Heisenberg interaction does not yield the CNOT operation directly, while

$$\exp\left[-i\mathcal{H}_{i,j}^{JJ}(E_{i,j}^{JJ})\frac{\pi}{E_{i,j}^{JJ}}\right] = e^{i\pi/4}\begin{pmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{pmatrix}$$

corresponds to the SWAP operation, which simply swaps two qubits. We denote it by the following symbol:

$$\text{SWAP} \equiv$$ 

We will consider the SWAP operation in more detail in the next section. Since the SWAP operation cannot entangle two qubits (although it is a genuine two-qubit gate!), one has to use alternative ways to produce CNOT, e.g., via the square root of SWAP (briefly denoted by $\sqrt{\text{SWAP}}$) which can be obtained by applying the Heisenberg Hamiltonian only for a time $0.5\,\pi\hbar/E_{i,j}^{JJ}$. Then, CNOT can be generated [LD98] via



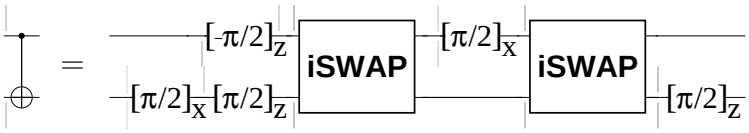
However, here $\sqrt{\text{SWAP}}$ has to be applied twice: the CNOT gate *cannot* be constructed by applying a $\mathcal{H}_{i,j}^{JJ}$-based gate only once [Mak00]. The converse is true for the construction of the SWAP operation using the ZZ interaction: while CNOT can be obtained in one step, SWAP requires two two-bit operations.

In [Mak00] it is also shown that the XY interaction Hamiltonian $\mathcal{H}_{i,j}^{XY}$ can *neither* generate the CNOT operation *nor* the SWAP operation by applying an XY-based gate only once. Nevertheless it is sufficient to build a CNOT gate. An appropriate "elementary" two-bit gate is the ISWAP operation which is obtained by applying $\mathcal{H}_{i,j}^{XY}(E_{i,j}^{XY})$ for a time $t = \pi\hbar/E_{i,j}^{XY}$:

$$\boxed{\text{iSWAP}} := \begin{pmatrix} 1 & & & \\ & 0 & i & \\ & i & 0 & \\ & & & 1 \end{pmatrix} = \exp\left[-i\mathcal{H}_{i,j}^{XY}(E_{i,j}^{XY})\frac{\pi}{E_{i,j}^{XY}}\right].$$

It has been noted before that this gate is useful in order to generate more complex quantum operations [SF01, KBDW01, EWD+01].

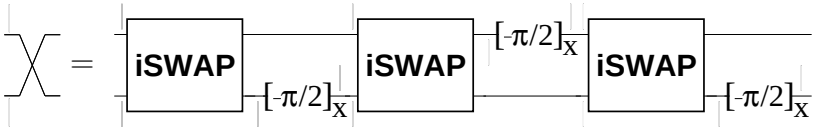
By applying the ISWAP gate twice, the CNOT operation can be constructed



. (4.2)

We mention that in complex circuits the length of this sequence can be reduced by noting that the "outer" one-bit operations partially cancel out with preceding or subsequent one-bit gates.

Of course, also the SWAP operation can be built with ISWAP gates, e.g., by the following sequence:



. (4.3)

We mention that there exists also a proposal how to build the CNOT gate by using $\sqrt{\text{ISWAP}}$ [IAB+99].

Recently, the close relation between the three types of Hamiltonians and the corresponding two-bit operations has been demonstrated rigorously [VHC02].

4.3 Connecting distant qubits

In the preceding section we showed how the CNOT gate can be obtained using various interaction types, including those we found for the Josephson charge qubits. Still, this gives us these interactions only for those qubits which are coupled physically. As pointed out in the hardware chapter, in some setups we will not be able to couple each qubit to more than a few other qubits due to physical restrictions. On the other hand, our universal set of operations has to contain CNOTs between *all* pairs of qubits.

This problem can be solved—at least formally—by showing that in fact it is sufficient to have CNOTs only between some pairs of qubits as long as all qubits are connected by some "path" of CNOTs. This is true, for instance, if all the qubits are arranged in a line or in a ring with only nearest neighbor coupling.

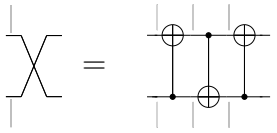
To this end, we reconsider the SWAP operation which already appeared in the previous section. The SWAP operation is of the form

$$\text{SWAP} = \begin{pmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{pmatrix} \equiv$$

,

mapping $(\alpha_A|0\rangle+\beta_A|1\rangle)(\alpha_B|0\rangle+\beta_B|1\rangle)$ to $(\alpha_B|0\rangle+\beta_B|1\rangle)(\alpha_A|0\rangle+\beta_A|1\rangle)$. Thus, the SWAP operation simply exchanges the two qubits A and B, or, to be more correct, their states. From this, it is also clear that the SWAP operation is not capable of entangling two qubits, although it is a real two-bit operation in the sense that it cannot be replaced by an outer product of local unitaries.

In the last section, we gave methods how to construct the SWAP gate for some of the mentioned interaction types. As we want to take a more formal point of view in this section, we note that since CNOT and local unitaries form a universal set of operations, there has to be a way how to to use them to construct SWAP. In fact, SWAP can be constructed using CNOT alone:



(4.4)

Now reconsider our problem: while we only have CNOT between some pairs of qubits, we would like to construct a CNOT between two arbitrary qubits A and B. Since our minimum requirement was that there is some connection between these two qubits, it follows that we can find a chain of coupled qubits, connecting A and B. In this chain, we start by swapping A and/or B with some qubit it is coupled to, using the sequence given above. By repeated swapping, we can make A and B nearest neighbors. Then, the CNOT operation between A and B is carried out and the swapping is reversed. This is illustrated in Fig. 4.1.

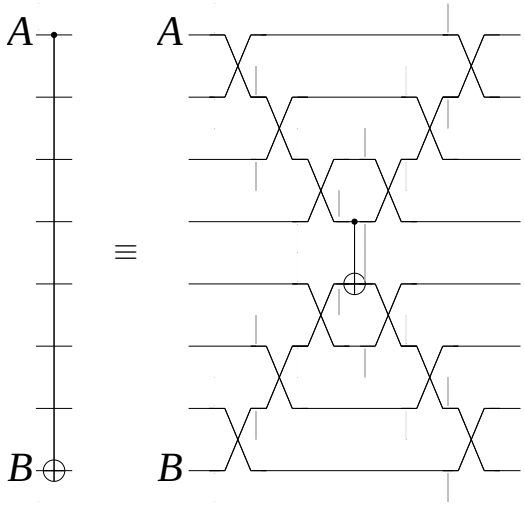


Figure 4.1: Coupling of distant qubits. Consider the following setup with 8 qubits and nearest neighbor coupling. By successive swappings, a CNOT between distant qubits can be accomplished. The SWAP operations can be constructed using CNOT according to (4.4).

Although this method gives us a tool to overcome restrictions in the coupling of the qubits, this workaround is rather unsatisfactory, since it will increase the number of operations considerably, especially in systems providing only nearest neighbor interaction. In fact, each operation could generate an overhead of as much as $O(N)$ operations in a system with N qubits.

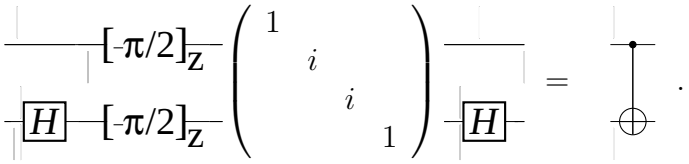
4.4 A natural gate for the XY interaction

In the following section, we reconsider the XY interaction. As we have seen, it can be used to generate a two-qubit gate which (together with single-bit rotations) is sufficient for universal quantum computation. However, neither CNOT nor SWAP can be realized directly by using the interaction part of the Hamiltonian only once, in contrast to the ZZ and the Heisenberg interaction. Now we ask whether there exists a "natural" two-qubit operation similar to CNOT also for the XY-interaction, i.e., a gate which can be viewed as the quantum case of a classical reversible operation like the CNOT or the SWAP gate.

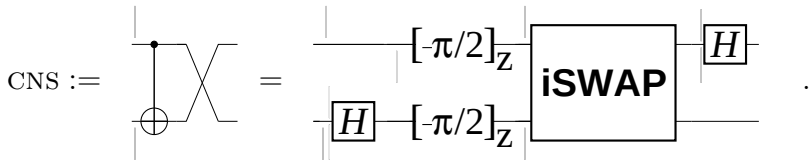
By analyzing the matrix of the ISWAP gate we see that it can be decomposed as

$$\text{ISWAP} = \begin{pmatrix} 1 & & & \\ & 0 & i & \\ & i & 0 & \\ & & & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ & i & & \\ & & i & \\ & & & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{pmatrix} .$$

The second matrix represents the SWAP operation, while the first matrix is equivalent to CNOT up to one-bit operations since



Thus it follows that the ISWAP gate is equivalent to a combination of CNOT and SWAP. The exact sequence is



For the sake of brevity, we introduce the name CNS ("CNOT+SWAP") for the new gate.

Remarkably this combined gate requires only a single operation using the coupling Hamiltonian and can therefore be regarded as a natural gate in the sense explained above.

One is tempted to object that the combination of CNOT and SWAP makes the CNS gate difficult to handle. While this is true in principle, one may notice that in the case of qubit couplings only between nearest neighbors it is necessary anyhow to swap the qubit states, as discussed before. Therefore, one can try to exploit this feature by rearranging the circuit in such a way that CNOT and SWAP operations appear together and can be replaced by a CNS operation. Moreover, it should be mentioned that the CNS operation is considerably shorter than both

the CNOT and the SWAP operation (realized with the XY coupling); so even with an overhead of two-bit operations compared to a "standard" circuit (which uses CNOT and SWAP) the rearrangement of the network may yield an advantage in terms of the operation time required for the whole sequence.

4.5 Applications of the CNS gate

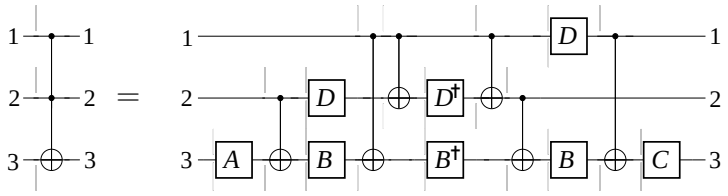
In this section we discuss two examples in order to demonstrate that the CNS gate derived above is surprisingly powerful in efficiently implementing quantum circuits in systems with nearest neighbor XY interactions. As a simple example we first discuss the Toffoli gate. In order to show that the method works for more complex networks as well, we then present an implementation of the five-bit error correction found by DiVincenzo and Shor [DS96].

We mention that the methods we present in this section work equally well for a couple of other networks. Similar solutions can be found, e.g., for the Quantum Fourier Transform (see, e.g., [CEMM98]), for the quantum adder described in [Dra00] (which is adding two quantum numbers), or for an adder which is adding one classical to one quantum number [Bea02].

4.5.1 The three-bit Toffoli gate

The three-bit Toffoli gate is the generalization of the CNOT gate with two control bits—it inverts the third bit if and only if the first two bits are in the $|1\rangle$ state. It is of special interest since it is the elementary gate for *classical* reversible computation. For this reason it often appears in circuits for tasks that also can be solved by classical reversible computers, e.g., the modular exponentiation used in Shor's factoring algorithm [VBE96].

There are various proposals to implement the Toffoli gate. The shortest one using the CNOT gate as the only two-bit gate is given in [DiV98] (which is a simplification of the version in [BBC$^+$95]) and involves six CNOT gates:



(4.5)

where

$$A = \begin{pmatrix} 1 & \\ & i \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1-\sqrt{2} \\ \sqrt{2}-1 & 1 \end{pmatrix},$$

$$C = \begin{pmatrix} 1 & \sqrt{2}-1 \\ i(\sqrt{2}-1) & -i \end{pmatrix}, \qquad D = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix},$$

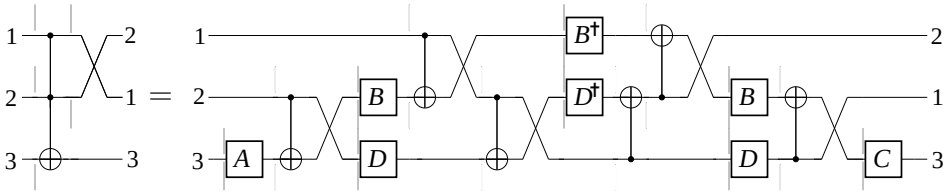
up to normalization factors.

We are considering systems with only nearest neighbor interaction. As the Toffoli gate often appears as an element in a circuit with more than three qubits, interaction will be possible only between the qubit pairs 1-2 and 2-3, but not between qubits 1 and 3. Since in the circuit (4.5) two of the CNOT gates act between qubits 1 and 3, one will have to swap one of the two qubits with qubit 2 in order to make qubits 1 and 3 nearest neighbors. This swapping has to be undone after the CNOT operation in order to bring the qubits back into the right order. Therefore one has to perform four SWAP operations in addition to the six CNOTs. The number of CNOT or ISWAP gates to build a SWAP gate is three. It turns out, however, that the CNOT operations between qubits 1 and 3 can be generated with only five (instead of seven) nearest neighbor CNOTs. Thus one ends up with

- $4 + 2 \cdot 5 = \mathbf{14}$ CNOT gates or
- $6 \cdot 2 + 4 \cdot 3 = \mathbf{24}$ ISWAP gates

required to obtain one Toffoli gate.

Now we rearrange the circuit to make use of the properties of the CNS gate. We have found the sequence



(4.6)

which replaces five CNOT operations by CNS operations and requires only a single additional SWAP operation (i.e., in addition to the sequence in (4.6)) to correct for the fact that the sequence does not exactly implement the Toffoli gate but rather exchanges the qubits 1 and 2 (as can be seen easily by retracing the three lines in the circuit above). For the separate SWAP, three ISWAPs have to be done according to (4.3). Thus, one finds that the total number of ISWAP gates needed to implement the Toffoli gate in a chain of qubits is

- $5 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 = \mathbf{10}$ ISWAP gates

compared to the 24 ISWAP gates required for the "naïve version" above.

Note that it is not necessarily a disadvantage to swap qubits 1 and 2 as in (4.6). Firstly, it can be regarded as a combination of the Toffoli gate and SWAP analogously to CNS. In fact, by replacing some of the CNOTs by CNS gates in the Toffoli network in (4.5) every possible permutation of the input bits can be achieved (amusingly, *except* the constant permutation), so one could try to exploit this in more complex circuits the same way as the fact that CNS is a combination of CNOT and SWAP.

Secondly, in quantum computing there are numerous circuits which first execute a sequence and then repeat the operations in the reverse order, e.g., to reset some ancilla qubits. In this case swapping of two qubits is often irrelevant (for examples of such circuits, see the network for the N-bit Toffoli gate in [BBC$^+$95] or the circuit of the quantum adder in [VBE96]).

4.5.2 A five-bit error correcting code

As a more advanced application where the use of CNS gates gives a considerable advantage over the simple "translation" of the network, we present an implementation of the five-bit error correction network which was found by DiVincenzo and Shor [DS96]. This network (see Fig. 4.2) can compensate arbitrary one-bit errors as long as they occur only in one of the encoding qubits at a time.

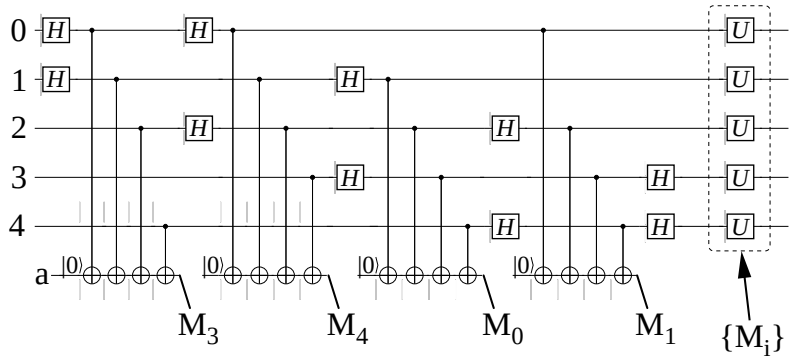


Figure 4.2: The five-bit error correction network as suggested by DiVincenzo and Shor [DS96]. The protected qubit state is encoded in the qubits 0 to 4, and the measurements M_i of the ancilla yield the error syndrome which determines the correction operations U. Instead of a single ancilla it is equally possible to use four different ancillae such that each measurement is performed on a (physically) different ancilla.

The protected qubit is encoded in five physical qubits 0 to 4 as a superposition of five-qubit states. The error correction network makes use of four ancilla bits which are initialized to $|0\rangle$ before the network is applied. After carrying out the sequence of operations, the ancillae are measured (in the standard basis).

While the implementation of this circuit appears rather hopeless if there is no direct interaction between the ancillae and *each* encoding qubit, we will show that the CNS gate makes a straightforward implementation of this network possible. To this end, we consider a setup of nine qubits (five qubits encoding the protected state plus four ancilla bits) arranged in a ring, i.e. we have nearest neighbor couplings with periodic boundary conditions.

By properly rearranging the gates, we obtain the circuit shown in Fig. 4.3. The labels 0 to 4 on the lines denote the five qubits of the error correcting code and, correspondingly, a_0, a_1, a_3 and a_4 are the labels of the four ancilla bits for

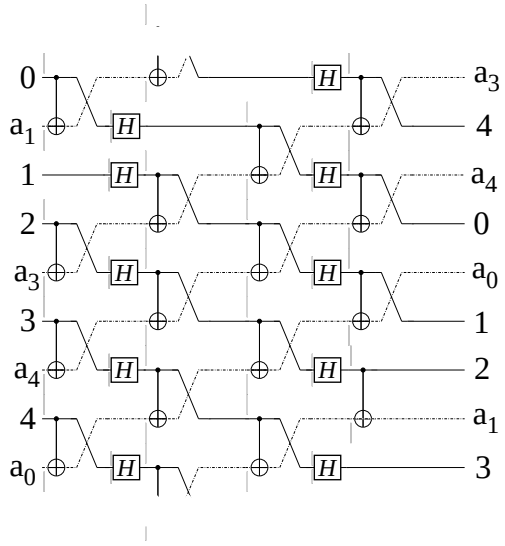


Figure 4.3: Implementation of the five-bit error correcting code with CNS gates (see text).

the M_is (see Fig. 4.2). After the application of the network, the four ancillae have to be measured and the corresponding operations on the five bits to correct the error syndrome have to be applied. Then the ancillae have to be reset to the $|0\rangle$ state. For the whole setup, only one separate CNOT is necessary, all other CNOT gates can be replaced by CNS gates. Separate SWAP operations are not required at all. One may notice that the two-bit operations in this network can be done in parallel which makes the execution of the whole sequence considerably faster.[1]

The equivalence of the networks in Fig. 4.2 and Fig. 4.3 is explained in detail in Section 4.5.3. Note that this implementation of the error correcting code leaves the qubits in the original order, but *rotates* them by three bits. Therefore, one has to keep track of the position of each bit. In any case, after three subsequent applications of the error correction scheme the encoding bits are back in their original order. As to the ancilla bits, their order is changed in a well-defined way. This has to be taken into account for the interpretation of the measurement outcome. After the measurement, the order of the ancillae is irrelevant since they are re-initialized to $|0\rangle$.

Until now, we have achieved an efficient implementation of the five-bit error correction code using the CNS gate as an elementary building block. Let us conclude this discussion by studying the implementation closer to the hardware level. In order to realize the network in a setup of Josephson charge qubits with SQUID loop coupling, i.e., nearest neighbor XY interaction, one can rewrite the

[1]Formally a parallel execution looks feasible also if the network can be implemented with physical couplings between arbitrary pairs of bits, in particular between the ancillae and *each* encoding qubit. Note, however, that in such schemes there is typically only *one* channel which mediates the coupling between the various qubits [IAB+99, MSS99]. Consequently, simultaneous execution of several two-qubit operations would result in N-qubit dynamics ($N > 2$) which is to be excluded. Therefore, in certain cases application of nearest neighbor coupling appears to be even more powerful than coupling between arbitrary pairs of qubits.

circuit in Fig. 4.3 in terms of ISWAP operations. The sequence can be simplified considerably, the hints are described in section 4.5.3. The result is illustrated in Fig. 4.4. If we assume that the energies in the one-bit and two-bit part of the Hamiltonian are of the order $E_{\rm typ}^{1-{\rm bit}}$ and $E_{\rm typ}^{2-{\rm bit}}$, respectively, the total operation time of the sequence (without measurement, correction step and resetting the ancillae) is $2.5\,\pi\hbar/E_{\rm typ}^{1-{\rm bit}} + 5\,\pi\hbar/E_{\rm typ}^{2-{\rm bit}}$.

We mention that efficient solutions for similar tasks in error correction have been developed also in [BS97, BLDS99].

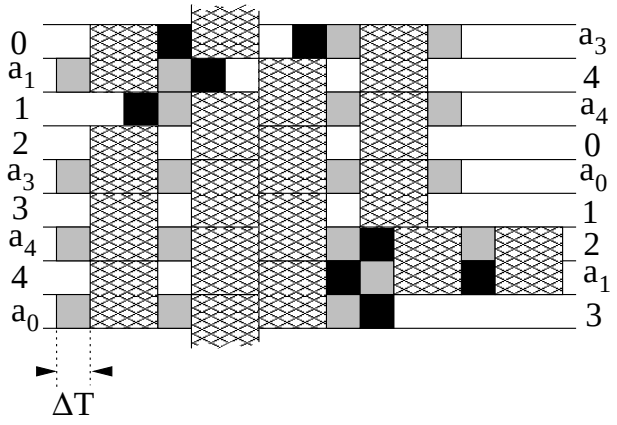


Figure 4.4: Sequence of operations for the five-bit error correcting code [DS96] for a Hamiltonian with controllable σ_x and σ_z part and nearest neighbor XY interaction between the qubits which are arranged in a ring. The horizontal direction in the figure shows the time axis while in the vertical direction, the various actions on each qubit are displayed. The boxes represent an active single-qubit or two-qubit gate, while white spaces denote idle periods. Black boxes correspond to a σ_z rotation, and gray boxes to a σ_x rotation. The big hatched boxes correspond to the action of an XY coupling term between two qubits. The time grid is $\Delta T = 0.5\,\pi\hbar/E_{\rm typ}$, where we have assumed equal energy scales for one-bit and two-bit Hamiltonians: $E_{\rm typ}^{1-{\rm bit}} \simeq E_{\rm typ}^{2-{\rm bit}} \simeq E_{\rm typ}$, which, e.g., is the case for Josephson charge qubits coupled by SQUID loops. The signs of the corresponding energies are not contained in the diagram. Two-bit blocks (representing ISWAP gates) correspond to CNS operations; the double ISWAP block between qubits 2 and a_1 at the end of the sequence represents the separate CNOT gate.

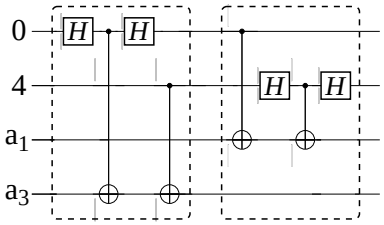
4.5.3 Appendix to the CNS examples

Equivalence of the error correcting networks

We need to show that the two error correction circuits in Fig. 4.2 and Fig. 4.3 are equivalent, i.e., that they correspond to the same unitary. To this end we

note that gates which have no bits in common commute trivially. Further, two subsequent CNOT operations commute if they act on the same target bit (in our case this means that they have the ancilla in common). Also, two CNOT gates with the same control bit and different targets commute as long as the control bit is not modified by a *single* Hadamard operation between them.

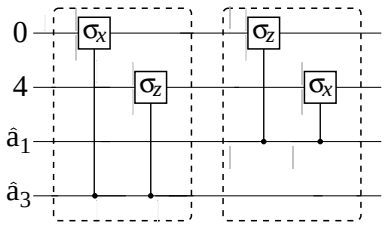
Starting from these observations, the question whether the two error correcting networks are identical reduces to proving that the two marked blocks in



(4.7)

do commute.

This can be seen as follows. First, choose the Hadamard transformed basis for the ancillae (we will denote them by $\hat{a}_i$). In this basis, the CNOT gates originally enclosed by Hadamard operations become CNOTs with control and target reversed, i.e., controlled σ_x operations. The other CNOT gates turn into controlled phase flips (i.e., controlled σ_z operations), analogously. Thus, the network in (4.7) corresponds to[2]



. (4.8)

In this representation, it becomes obvious that the two blocks indeed do commute: if at least one $\hat{a}_i$ is zero, one of the two controlled operations on each qubit 0 and 4 is the identity, and the operations commute. On the other hand, if $\hat{a}_1 = \hat{a}_3 = 1$, exchanging σ_x and σ_z on *one* bit results in a global minus sign. Changing the order of the two blocks in (4.8) corresponds to two simultaneous changes of this kind and leaves the all-over result unchanged. Therefore, the networks in Fig. 4.2 and Fig. 4.3 are identical.

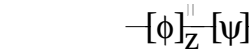
Simplifications for circuits with ISWAP

We now briefly describe the methods which can be used for the simplification of a network like the one presented for the five-bit error correction circuit in

[2]The newly introduced symbols are controlled-U operations. Similar to CNOT, the qubit marked with a dot is the control qubit, while the box on the target qubit contains an one-bit operation U. Depending on the state of the control bit, I (control bit is $|0\rangle$) or U (control bit is $|1\rangle$) is applied to the target bit.

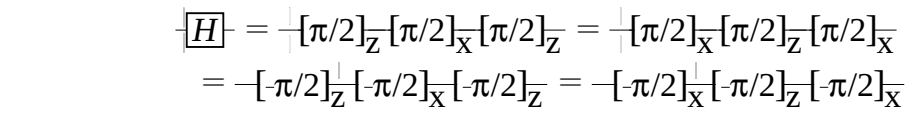
Section 4.5.2, cf. Fig. 4.4. We will consider rotations about the x and the z-axis, and the ISWAP operation as the basic building blocks for the implementation. All other operations will be expressed in terms of these operations. Although the application of the ideas is rather straightforward it is difficult to provide formal recipes how to use them.

1. One-bit simplifications. First, rotations about the same axis can be collected, e.g.:



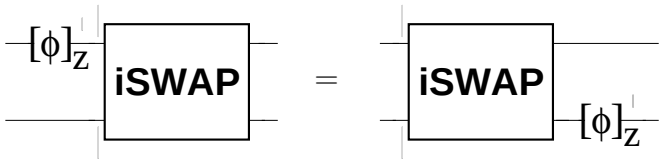
Clearly, rotations by an angle of 2π can be dropped. Operation time can be saved by applying a $-\pi/2$ rotation instead of a $+3\pi/2$ one.

A more sophisticated problem is the simplification of compound expressions of x *and* z rotations. To this end, the following ways to express the Hadamard transformation are useful:



By choosing the appropriate way to represent the Hadamard transformations in the network (or by sometimes "artificially" creating one of these triples, for example by inserting a pair of operations whose product is unity) and replacing it by another one, considerable simplifications can be achieved. The applicability of these simplifications becomes particularly apparent if the single-bit operations are considered in the context of two-bit operations.

2. Two-bit simplifications. There is essentially only one way to simplify two-bit expressions for circuits containing ISWAP:



i.e., z rotations "commute" with ISWAP if simultaneosly the z rotation is flipped to the other qubit.

3. Ancilla simplifications. Finally, one can apply also one-bit simplifications to the ancilla bits which are possible due to the fact that we know the initial state of the ancilla and, moreover, the ancillae are measured in the $\{|0\rangle, |1\rangle\}$ basis at the end.

At the beginning of the error correction sequence, the ancillae are set to $|0\rangle$. Therefore, z rotations immediately after the initialization can be omitted (since global phases are not important).

Further, let us assume the ancilla is in the state $a|0\rangle + b|1\rangle$ just before the measurement. As the measurement is performed in the $\{|0\rangle, |1\rangle\}$ basis, a z rotation before the measurement would not affect the result. Therefore, also these z rotations need not be considered.

Chapter 5

Implementation of the Deutsch–Jozsa algorithm

In this chapter, we show how the Deutsch–Jozsa algorithm for an arbitrary number of qubits can be implemented on the hardware setup described in Chapter 3. It therefore contains the central results of the thesis.

The chapter is structured as follows. As an introduction, we motivate the interest in implementations of the Deutsch–Jozsa algorithm and show that the central issue is the implementation of the oracle. We then consider the existing (three-bit) implementations of the algorithm and point out the fundamental principles common to all of them. We discuss several possibilities how to extend these principles to more than three qubits, and demonstrate that each of these approaches leads to problems for higher qubit numbers.

This motivates the quest for a universal implementation, which works the same way for any oracle and any qubit number. We derive such an implementation which can be programmed easily to realize any function under consideration. We start by presenting the general concept, and continue with different ideas how to implement the corresponding network. We show that these programmable networks can also be used for other tasks, and discuss some examples. The chapter closes with a discussion of the complexity of these networks.

Preliminary results of this chapter have been published in [SS02a]. Another article is in preparation.

5.1 Introductory considerations

5.1.1 Motivation

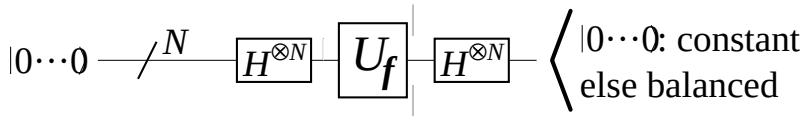
In this chapter, we discuss the implementation of the Deutsch–Jozsa algorithm. What is the motivation to do that? As we have seen in Chapter 2, the Deutsch–Jozsa algorithm is mainly a *theoretical* construction, intended to demonstrate

that quantum computers can outperform classical computers in terms of oracle query compexity. In this sense, one could argue that an implementation of the Deutsch–Jozsa algorithm is of purely theoretical interest.

On the other hand, the Deutsch–Jozsa algorithm is the algorithm which has been implemented (or for which implementations have been proposed) most often, and for most physical realizations. A first reason is that among all known quantum algorithms, the Deutsch–Jozsa algorithm is the most simple. Moreover, it displays the key features of quantum computation, parallelism and entanglement. We concentrate on the Collins–Kim–Holton version (see Section 2.2.1, pg. 18 et seqq., or [CKH98]) of the Deutsch–Jozsa algorithm without the auxiliary qubit, since this bit does not get entangled with the other qubits by the algorithm. In the sense mentioned above, the implementation of the Deutsch–Jozsa algorithm is a meaningful test for the feasibility of quantum computation with a given hardware setup.

5.1.2 The key issue is the implementation of the oracle

In the following, we investigate the implementation of the Deutsch oracle on Josephson charge qubits. Therefore, recall the main steps of the Deutsch–Jozsa algorithm as derived in Section 2.2.1:



First, the N-qubit register has to be prepared in the $|0\cdots0\rangle$ state. We mentioned in Section 3.1 that there exist schemes how to initialize each qubit of the register to some well-defined value. We also showed in Section 4.1 how to do arbitrary one-bit operations with the Hamiltonian provided by our hardware. This allows us to initialize each qubit to the $|0\rangle$ state. Also, the two N-bit Hadamard operations can be performed using our Hamiltonian, since they can be written as products of one-bit Hadamards. We also mentioned that there exist schemes how to measure the state of a charge qubit.

Thus, the only task which does not have an obvious solution is the implementation of the oracle. Recall that we want to implement the Deutsch–Jozsa algorithm for all possible oracles in order to show the feasibility of quantum computing in general and not only in some specially chosen cases.

At first glance, the implementation of the oracle might not look like a serious obstacle. But in fact, it is very complicated. To give an idea why this is the case, we estimate the number of different oracles which have to be implemented.

On the one hand, there are the oracles belonging to the two constant functions $f \equiv 0$ and $f \equiv 1$. The implementation of these oracles is most easy—the corresponding matrix is an identity matrix up to an irrelevant global phase. On

$B(1)$=2	$B(4)$=12870	$B(7)$=$2.40 \cdot 10^{37}$	$B(10)$=$4.48 \cdot 10^{306}$
$B(2)$=6	$B(5)$=$6.01 \cdot 10^{8}$	$B(8)$=$5.77 \cdot 10^{75}$	$B(11)$=$5.70 \cdot 10^{614}$
$B(3)$=70	$B(6)$=$1.83 \cdot 10^{18}$	$B(9)$=$4.73 \cdot 10^{152}$	$B(12)$=$1.30 \cdot 10^{1231}$

Table 5.1: Number of balanced function $B(N)$ as a function of the number of qubits.

the other hand, all the balanced functions have to be implemented. The number of balanced functions $B(N)$ is

$$B(N) = \begin{pmatrix} 2^N \\ 2^{N-1} \end{pmatrix} ,$$

since $f : \{0,1\}^N \to \{0,1\}$ acts on a space with 2^N elements, and half of the function values is 1. For the first few N, the number of balanced functions $B(N)$ is given in Table 5.1. The asyptotic behavior of $B(N)$ can be determined using Stirling's formula. For large N, one finds

$$\begin{aligned} B(N) &\approx \frac{1}{\sqrt{2\pi}} \frac{1}{2^{N/2-1}} 2^{2^N} \\ &\propto \frac{2^{2^N}}{\sqrt{2^N}} \\ &\approx 2^{2^N} . \end{aligned}$$

This means that the number of oracles grows almost doubly exponentially[1] with N, thus making it hard to give implementations for all different oracles without finding some underlying principles. The discussion of existing approaches to this, and the development of a new unified method, are subject of this chapter.

5.2 Existing implementations of the Deutsch–Jozsa algorithm

Up to now, several implementations of the Deutsch–Jozsa algorithm have been worked out. There exist implementations for up to two qubits in the classical version with the auxiliary bit [CY95, LBF98, JM98, MDAK01, CMT01, KY02], and there exist some implementations for three qubits in the Collins version without an auxiliary qubit [CKH$^+$00, KLLC00, SF01, VAZ$^+$01, ADK01, BGLA02]. In the following, we focus on the three-bit solutions. They are of special interest to us since—as discussed in Chapter 2—the three-bit case is the first to produce

[1] This can be seen nicely in Table 5.1—the exponent of $B(N)$ is growing exponentially.

entanglement, and since for three qubits for the first time a classification of the oracles proves useful.

Still, we omit some implementations from our discussion—for good reasons, though. Firstly, there exists an experiment for the *four*-bit Deutsch–Jozsa algorithm plus auxiliary qubit [MFM$^+$00]. But in this experiment, only one single balanced function was implemented, which did not even involve entanglement.

Then, there exist two implementations of the three-bit Deutsch–Jozsa algorithm which use molecular vibronic states as quantum registers [VAZ$^+$01, BGLA02]. We do not consider these proposals, since the underlying concept of quantum computation is quite different, as discussed in [VAZ$^+$01]. Basically, those implementations do not use 3 quantum bits but *one* 8-level system. Of cource, the physics is the same, since in both cases the Hilbert space is 8-dimensional. But there no longer exist concepts like one-bit operations or two-bit operations, such that the notion of universal quantum computation—which is a central issue in the proof of computational speed-up—no longer applies. Rather, every operation has to be tailored by hand. Also, the system is not scalable, since the number of different vibronic levels in the molecule which can be controlled properly is limited to some hundred. We mention that recently a qubit-based concept for quantum computation with rotovibrational states of molecules has been proposed [TdVR02].

Finally, there also exists an implementation of Deutsch–Jozsa using an optical scheme for quantum computation with *only one* photon [Tak00], where the state of the register is determined by the path the photon takes (which, of course, allows for superpositions as well). But neither this proposal is qubit-based, since, e.g., applying a Hadamard to *one* bit requires beam splitters between all pairs of beams. Of course, there also exist qubit-based proposals for quantum computing with photons.

5.2.1 Classification by operation sequences

Before going into the details of the remaining three-bit implementations, let us recall the basic things about the oracle in Deutsch's algorithm. We consider the oracles of balanced functions, i.e., functions with $f(\mathbf{x}) = 1$ for exactly half of all $\mathbf{x}$. The oracle itself maps

$$U_f : |\mathbf{x}\rangle \mapsto (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle \ ,$$

i.e., it can be represented by a diagonal matrix, where the entries are ± 1. The phase of a state $|\mathbf{x}\rangle$ is flipped exactly if $f(\mathbf{x}) = 1$.

Classification by Siewert and Fazio

We will focus our discussion on the proposal given by Siewert and Fazio for Josephson charge qubits [SF01], and then compare the generalized results to the

other implementations. The hardware used consists of three Josephson charge qubits coupled by SQUID loops, i.e., the coupling is of the XY type. The coupling is available between all pairs of qubits, which in fact is the simplest case of qubits with nearest neighbor coupling arranged in a ring.

The authors start by noting that—due to the irrelevance of a global phase of π—only 35 of the 70 balanced functions have to be considered. They classify these 35 functions according to the separability of the oracle. There are 7 balanced functions where the oracle can be decomposed into a product of one-bit operations (class I), and 12 more functions where the oracle factorizes into an one-qubit and a two-qubit part (class II). The remaining 16 gates cannot be factorized. Still, the authors note that this fully entangling class in fact consists of two somehow different classes (class III, IV).

Then, they show how these four classes can be implemented. For the fully separable oracles of class I, no two-bit operations are necessary—one only needs diagonal one-bit (i.e., phase-shifting) operations. For the partially entangling class II, one two-bit operation is necessary. For the fully entangling oracles, finally, *at least* two entangling operations are necessary. It turns out that this is the criterion to discriminate the two subclasses. For class III, two two-bit operations are needed, while for class IV, three of them are necessary.

Classification by controlled phase flips

We thus see that the true classification criterion for the oracles is the number of two-qubit operations required for the implementation (i.e., the operation sequence modulo local unitaries) rather than the separability.

We define the two-bit controlled phase flip ${}^2\mathrm{CPF}$ as

$$ {}^2\mathrm{CPF} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1 \end{pmatrix} , $$

i.e., the operation which flips the phase of the $|11\rangle$ state. More generally, an N-bit controlled phase flip ${}^N\mathrm{CPF}$ is the operation which flips the phase of the $|1\cdots1\rangle$ state. This ${}^2\mathrm{CPF}$ and "one-bit controlled phase flips" ${}^1\mathrm{CPF} \equiv \sigma_z$ are the only operations needed to assemble the Deutsch oracle. Since they are both diagonal, all operations commute, and the operation sequence can be classified—up to one-bit unitaries—by the number of ${}^2\mathrm{CPF}$s, thus yielding exactly the classes we had before. (A proof of this is given in the next section.)

As an example, take the balanced function

$$ f_0 \equiv (f_0(x))_x = (0,0,1,0,1,1,1,0) . $$

The oracle of this function can be decomposed as follows (we use double square brackets $[\![\cdot]\!]$ for diagonal matrices, since otherwise the equation would not fit onto

the page):

$$U_{f_0} = \left[\!\!\left[\begin{array}{r} 1 \\ 1 \\ -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ 1 \end{array} \right]\!\!\right] = \underbrace{\left[\!\!\left[\begin{array}{r} 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \end{array} \right]\!\!\right]}_{=:A} \cdot \underbrace{\left[\!\!\left[\begin{array}{r} 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{array} \right]\!\!\right]}_{=:B} \cdot \underbrace{\left[\!\!\left[\begin{array}{r} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \end{array} \right]\!\!\right]}_{=:C} .$$

The matrices A, B, and C on the right hand side can be decomposed as

$$A = \begin{pmatrix} 1 & \\ & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & \\ & -1 \end{pmatrix} \otimes I = {}^1\mathrm{CPF} \otimes {}^1\mathrm{CPF} \otimes I \ , \tag{5.1}$$

$$B = I \otimes \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1 \end{pmatrix} = I \otimes {}^2\mathrm{CPF} \ , \tag{5.2}$$

$$C = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1 \end{pmatrix} \otimes I = {}^2\mathrm{CPF} \otimes I \ . \tag{5.3}$$

This means that f_0 belongs to the fully entangling oracles, and within the fully entangling oracles to class III, the one with only two ^{2}CPFs appearing in the operation sequence (in contrast to class IV, where three ^{2}CPFs appear).

Since the setup for the three-qubit case provides coupling between all pairs of qubits, and the ^{k}CPFs commute (they are all diagonal!), all sequences containing an equal number of ^{2}CPFs are equivalent up to permutations of the qubits, and up to one-bit operations. This implies that each of the classes consists of all the oracles which have the same operation sequence modulo these transformations.

5.2.2 Classification by monomials

Before proceeding to the extension of this approach to four and more qubits, we derive a more formal description of the classification scheme discussed above. Although it looks a bit tedious at a first glance, this scheme provides an easy and straightforward method to determine the class a balanced function belongs to.

We explain the method by means of an example. Take the function analyzed in the last section, $f_0 \equiv (0,0,1,0,1,1,1,0)$, and recall that the oracle U_{f_0} is defined on the computational basis by

$$U_{f_0} : |\mathbf{x}\rangle \mapsto (-1)^{f_0(\mathbf{x})} |\mathbf{x}\rangle \ . \tag{5.4}$$

We start by rewriting f_0 as a sum of Kronecker deltas,

$$\delta_{\mathbf{x},\mathbf{y}} = \begin{cases} 0 \text{ if } \mathbf{x} \neq \mathbf{y} \\ 1 \text{ if } \mathbf{x} = \mathbf{y} \end{cases} :$$

$$\begin{aligned} f_0(\mathbf{x}) &= \sum_{\mathbf{y}:f(\mathbf{y})=1} \delta_{\mathbf{x},\mathbf{y}} \\ &= \delta_{010,\mathbf{x}} + \delta_{100,\mathbf{x}} + \delta_{101,\mathbf{x}} + \delta_{110,\mathbf{x}} \ . \end{aligned} \tag{5.5}$$

Each of the δs can be expanded as a polynomial in (x_i), e.g.:

$$\delta_{010,\mathbf{x}} = (1-x_1)x_2(1-x_3) = x_2 - x_1x_2 - x_2x_3 + x_1x_2x_3 \ .$$

This works since $x_i \in \{0,1\}$, and therefore $(1-x_1)$ equals to $\text{NOT}(x_i)$. Furthermore, a product corresponds to an AND gating of the factors, which altogether yields exactly the Kronecker delta.

By rewriting all four δs as polynomials in (x_i)—note that each of the polynomials contains $\pm x_1x_2x_3$—and by substituting the polynomials into (5.5), f_0 can be rewritten as

$$f_0(\mathbf{x}) = x_1 + x_2 - x_1x_2 - x_2x_3 \ .$$

Using (5.4), we see that the oracle does

$$\begin{aligned} U_{f_0} : |\mathbf{x}\rangle &\mapsto (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle \\ &= \underbrace{(-1)^{x_1}(-1)^{x_2}}_{^1\text{CPFs}} \underbrace{(-1)^{x_1x_2}(-1)^{x_2x_3}}_{^2\text{CPFs}} |\mathbf{x}\rangle \ . \end{aligned}$$

In this representation, the operation sequence can be seen clearly! The first two factors correspond to two σ_z, i.e., ^{1}CPF, operations on qubits 1 and 2, cf. (5.1). The third and fourth part correspond to one ^{2}CPF each, namely the ones given in (5.2) and (5.3). Thus, each monomial corresponds to some controlled phase flip ^{k}CPF, where the number of x_is in the monomial is the number k of bits the ^{k}CPF operates on.

It is important to note that, since the polynomial expansion of *each* δ contains $\pm x_1x_2x_3$, and for a balanced function the number of δs is $2^N/2 = 4$, i.e., even, $(-1)^{x_1x_2x_3}$ cannot appear in the oracle. Thus, the only nonlocal operations are the ones with the monomials x_1x_2, x_1x_3 and x_2x_3, leading to the classification discussed—class I–IV contains 0–3 of these monomials, respectively.

Another interesting observation is due to the fact that we are not interested in the exact polynomial for f_0 but in the action of the oracle U_{f_0}. Since the oracle uses only $(-1)^{f_0}$, f_0 (and thus its polynomial) only has to be taken modulo 2. Therefore, in the polynomial expansion of f_0, each monomial may or may not appear, but effectively it cannot have a prefactor other than 0 or 1. Alternatively,

one can replace the + in the polynomial by a $\oplus$, the sum modulo 2. Also, since $x_i^2 = x_i$ (as $x_i \in \{0, 1\}$), the number of monomials is limited to 2^N—each x_i may appear in the monomial (this includes the constant monomial 1 which leads to a global phase).

It is clear that this decomposition can be done for the oracle of *any* balanced function—in fact, for any function—and for an arbitrary number N of qubits. In any case, the highest order controlled phase flip, i.e., $|\mathbf{x}\rangle \mapsto (-1)^{x_1 \cdots x_N} |\mathbf{x}\rangle$, vanishes for all balanced functions. This means that for the N-bit Deutsch oracle, controlled phase flips ${}^1\mathrm{CPF}$ up to ${}^{N-1}\mathrm{CPF}$ are sufficient—and also necessary, of course. This can be seen most easily by observing that $\sigma_z \otimes {}^{N-1}\mathrm{CPF}$ is the oracle of a balanced function.

Starting from this representation of balanced functions and their oracles, we will try to extend the classification scheme described above to more qubits in a meaningful way. Before we do that, though, we briefly compare these results to the other existing implementations.

5.2.3 Other implementation proposals

In the following, we briefly compare the other existing three-bit implementations with the classification discussed above.

Kim *et al.*

Firstly, we discuss the implementation by Kim *et al.* [KLLC00]. The authors show how the refined (i.e., Collins style) Deutsch–Jozsa algorithm can be implemented on a three-bit NMR quantum computer, and give experimental results carried out with ${}^{13}\mathrm{C}$ nuclear spins of 99% carbon-13 labeled alanine ($\mathrm{CH_3CH(NH_2)CO_2H}$) in $\mathrm{D_2O}$ solvent.

Searching for operation sequences, they also arrive at a classification according to the entangling operations. They note that some balanced functions require three-qubit interactions, but these can be simulated by two-qubit gates [KLL00]. Thus, they end up with four different classes of oracles, which are defined exactly the same way as the classes we derived before, namely by the number of two-bit operations needed to generate them.

Arvind, Dorai, and Kumar

Arvind, Dorai, and Kumar [ADK01] give an implementation of the Deutsch–Jozsa algorithm for up to three qubits in the Collins version on an NMR quantum computer. They mention that for up to two qubits no entanglement is involved, so that the three-bit case is the first meaningful test for quantum computation. For the three-qubit case, they give experimental results for an implementation with 5-nitro-2-furaldehyde.

Unfortunately, the authors only discuss (and perform) the implementation for 9 different functions, one of which is constant. Of the other 8 functions, 4 functions do not involve entanglement, 3 functions only entangle two qubits, and only one function entangles all three qubits. For the 3 partially entangling functions, they construct only one sequence, since the oracles can be transformed into each other by permutation of the qubits. Finally, for the one fully entangling function, simply an operation sequence is given, without any further discussions about how other oracles involving full entanglement could be created.

Collins *et al.*

Finally, there exists an NMR implementation of the Deutsch–Jozsa algorithm by Collins *et al.* [CKH$^+$00]. The authors start by mentioning that each function f (since it is a mapping of a finite set) can be represented by a polynomial in the ring $\mathbb{Z}_2$ (i.e., the ring of natural numbers modulo 2). These polynomials are obviously of the same form as the ones we derived using the δ representation.

They argue that for a demonstration of the feasibility of the Deutsch–Jozsa algorithm, in principle an implementation of all different balanced functions would be necessary. On the other hand, certain oracles can be transformed into each other by permutation of the qubits (note that all qubits are coupled). Therefore, the implementation has to be demonstrated for only one member of each permutationally equivalent group of functions. This leads to 10 different classes, since in this case also one-bit operations can make a difference. The first three classes are non-entangling (our class I), the second three are partially entangling (our class II), and the last four are fully entangling. Of these four, two belong to class III, the other two to class IV. The authors give pulse sequences for each type of oracle, and by permutation of the indices, the implementation of any oracle can be derived.

5.3 Extending the existing implementations

This section is devoted to the extension of the existing solutions. As we have seen, the analysis of the polynomial representation of the functions is very useful and can be shown to be the principle underlying all implementations. Therefore we will try to give some ideas how the existing classifications can be extended. To this end, the main issue is to figure out what the aim of an implementation and of a classification should be. As a result of the discussion, it will turn out that each of the suggested extensions has certain disadvantages, and the classification gets more and more complicated. Therefore, a universal implementation becomes desirable.

5.3.1 First idea: Just find the operation sequence

The minimum requirement for an implementation of the Deutsch oracle is to give an operation sequence how to perform the oracle. This, however, would not be a challange by itself. For example, we simple could rewrite f as a sum of Kronecker deltas, and implement each δ by an N-bit controlled phase flip ${}^N\mathrm{CPF}$.[2] But N-bit controlled phase flips are hard to generate, especially in systems with restricted couplings. Just think about the construction for the three-bit Toffoli gate in Section 4.5.1, which is equivalent to a ${}^3\mathrm{CPF}$, and imagine how this gets more and more complicated for higher qubit numbers.

Additionally, constructions of this kind can be simplified considerably. Consider, for instance, two N-bit controlled phase flips, i.e., δs, with only one bit negated relative to each other. Such a construction could be simplified to only one ${}^{N-1}\mathrm{CPF}$, see Fig. 5.1.

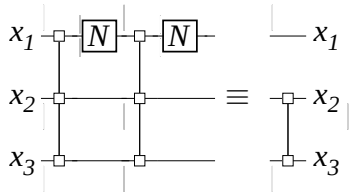


Figure 5.1: Simple equivalence which can be used to simplify circuits with controlled phase flips. The connected qubits with the squared boxes denote controlled phase flips $|x_1 \dots x_N\rangle \mapsto (-1)^{x_1 \cdots x_N} |x_1 \dots x_N\rangle$, thus symbolizing their symmetry against the permutation of qubits, and the boxed N is the NOT gate, i.e., σ_x.

There exist more possible simplifications of this kind. Additionally, one might feel tempted to find optimized operation sequences with respect to the specific hardware implementation, as we did for the Toffoli implementation of Section 4.5.1. But these simplifications would have to be carried out for each oracle unless we have classes. A classification would reduce the effort spent on simplification to the optimization of *one* function per class.

Simplifications like the ones mentioned above can also be carried out in the polynominal representation of the oracle. For the example in Fig. 5.1:

$$\begin{aligned} |\mathbf{x}\rangle &\mapsto (-1)^{x_1x_2x_3}(-1)^{(1-x_1)x_2x_3}|\mathbf{x}\rangle \\ &= (-1)^{x_1x_2x_3}(-1)^{x_2x_3}(-1)^{x_1x_2x_3}|\mathbf{x}\rangle \\ &= (-1)^{x_2x_3}|\mathbf{x}\rangle \ . \end{aligned}$$

[2] The implementation of the proper δ is obtained by applying NOT operations before and after the application of the ${}^N\mathrm{CPF}$ to all bits which are $|0\rangle$ in the δ. The ${}^N\mathrm{CPF}$ itself can be obtained by Hadamard-transforming the target bit of an N-bit controlled-NOT, for which constructions are known [BBC+95].

Obviously, the polynomial of an oracle can be rewritten in different ways. In the example above, e.g., we used $(1-x_1)x_2x_3 + x_1x_2x_3 = x_2x_3$. Some other possible rearrangements are

$$\begin{aligned} x_1x_2\cdots x_k \oplus x_2\cdots x_k &= (1\oplus x_1)x_2\cdots x_k \\ x_1x_3\cdots x_k \oplus x_2x_3\cdots x_k &= (x_1\oplus x_2)x_3\cdots x_k\ . \end{aligned}$$

The first equation simplifies one ^{k}CPF and one ${}^{k-1}$CPF to only one ^{k}CPF. The second equation tells us that we can replace the *two* ${}^{k-1}$CPFs by *one* ${}^{k-1}$CPF, plus two normal CNOTs. The CNOTs act on qubits 1 and 2, thus creating $x_1 \oplus x_2$ for the ${}^{k-1}$CPF and undoing it afterwards.

It is clear that a number of more or less sophisticated transformations, following similar ideas as these two, exist. Moreover, if we again consider our hardware setup with restricted interactions, a transformation might be useful or not depending on the qubits involved. This makes the transformations even more difficult to handle, since it is not clear *a priori* what a good simplification is.

5.3.2 Better idea: Classify the oracles

We have seen that in case we want to have good, i.e., optimized, operation sequences, we have to spend a lot of work on the optimization of each oracle separately. Therefore, we would like to reduce the effort of optimization to the simplification of only a few functions, and transfer the results to a whole class.

Classification up to one-bit operations

The simplest idea is to classify the oracles up to one-bit operations.[3] This means that two different balanced functions are regarded equivalent exactly if the corresponding oracles can be transformed into each other by local operations. For balanced functions, it is sufficient to allow σ_z as a local operation. It appears sensible to classify the oracles this way since local operations are easy to perform, so that they can safely be disregarded when optimizing the circuit. The local operations themselves, or the classification, can be found by analyzing the polynomial representation—the local $\sigma_z \equiv {}^{1}$CPFs are exactly the monomials with only one x_i. Classifying the 12870 4-bit Deutsch oracles this way yields 996 different classes, though. Since the number of oracles goes as 2^{2^N} while classification up to one-bit operations can identify a maximum of 2^N balanced functions, the number of classes is about $2^{2^N}/2^N$, which still is double exponential in N.

In fact, the classification up to one-bit operations can be extended. It is also possible to transform two oracles into each other by negating certain qubits before and after the application of the oracle. This leads to a further reduction

[3] An even simpler idea is to identify the two functions which only differ by a global phase of π. This gives a factor of two which is already included in all results which follow.

	one-bit operations		
symmetry	none	σ_z	σ_z + NOT
full	365	86	25
ring	1446	253	57
line	3255	526	88

Table 5.2: Number of classes for the four-bit Deutsch oracle, where the classification is done with respect to invariances under the system symmetry (i.e., permutations of qubits), as well as local σ_z and NOT operations.

of the number of classes (although it only gives another factor of 2^N), but the classification can no longer be performed by simply looking at the polynomial. Combining both criteria, we still get a total of 156 classes for 4 qubits (this criterion alone gives 555 classes).

Classification up to system symmetry

In fact, the classification can be improved even more. By exploiting the system symmetry, we find that certain classes can be implemented by exactly the same operation sequence, up to permutations of the qubits. Still, the allowed permutations depend on the symmetry of the system. If all qubits are coupled any permutation is allowed and we get a real advantage by that classification. On the other hand, in case of a ring with nearest neighbor coupling we can only apply rotations and a reflection (i.e., $x_i \leftrightarrow x_{N-i}$ for N qubits), and for a linear chain, just the reflection remains. In case of a ring this gives a factor of N. On the other hand, the classification might become quite hard, since we have to find a way to identify two different oracles up to the allowed permutations. Table 5.2 shows the number of classes for different symmetries of the setup, with and without one-bit invariances. Clearly, for five qubits the number of classes will be a lot larger. We do not give any values for this case, since the five-bit classification is very hard to perform with a computer, as the number of oracles is about 10^8 compared to 10^4 for four qubits (cf. Table 5.1).

Classification up to two-bit operations?

Some more sophisticted classification schemes might be considered. One could try, e.g., to define equivalence classes up to two-bit CPFs, since these are easy to assemble. But all classification concepts which only classify up to operations with a limited qubit depth only defer the problem—for some higher qubit number, we run into exactly the same trouble.

Thus, even if we try to classify the oracles, we get an exceedingly growing number of classes and a more and more complex classification scheme. For each

of the different classes—their number is growing almost double exponentially—we have to optimize an operation sequence which itself will be about exponentially long in most cases. This does not look like a satisfactory solution at all. Therefore, a universal implementation of *all* Deutsch oracles on the same footing—without referring to *any* type of classification scheme—becomes desireable.

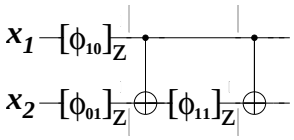
5.4 Programmable networks for the oracle

The following section presents the central result of this thesis. We derive a type of universal network which allows for the implementation of *all* Deutsch oracles on the same footing. The operation sequence is fixed, and the desired oracle can be programmed by simply adjusting the rotation angles of some well-defined z rotations. Therefore, such a network might well be considered as a black box (the fixed operation sequence of the network) with some knobs on it (the rotation angles), which are used to programme the black box. The box takes an N-qubit state as an input, and returns another N-qubit state. By properly adjusting the knobs on the black box one can programme the network to encode any desired Deutsch oracle. The connection between the balanced function and the knob positions is straightforward.

The section is divided into three parts. Firstly, we present the basic idea of the programmable networks without thinking about the physical implementation. In the second part, we discuss different possible implementations of these networks. Finally, we discuss two works which deal with similar ideas and compare them to our results.

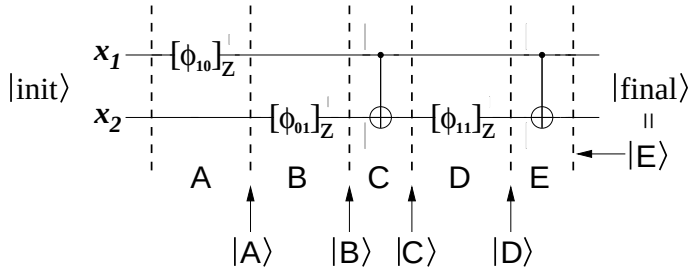
5.4.1 Programmable networks

To motivate the derivation of the network, we start by considering the following simple two-qubit setup:



. (5.6)

In order to facilitate the dicussion, we structure the network into blocks A to E, where each block contains only one operation.



. (5.7)

Introductory considerations

Clearly, it is sufficient to analyze the action of the network on initial states $|\text{init}\rangle$ chosen from the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, since we are concerned with linear operations. In order to find out how the network (5.7) acts upon a basis state, we need to consider the effect of the constituents of the network—z rotations and the CNOT gate.

Let us first consider the effect of $-[\phi]_{\mathsf{Z}}-$. By definition (cf. (2.1)),

$$-[\phi]_{\mathsf{Z}}- = \begin{pmatrix} e^{-i\phi/2} & \\ & e^{i\phi/2} \end{pmatrix} ,$$

so that the basis states are mapped as

$$-[\phi]_{\mathsf{Z}}- : |x\rangle \mapsto \exp\left[-i\tfrac{\phi}{2}(-1)^x\right]|x\rangle .$$

Therefore, $-[\phi]_{\mathsf{Z}}-$ does not mix the basis states at all (no off-diagonal matrix elements), but it rather introduces a *relative phase* of $\exp[-i\phi]$ between $|0\rangle$ and $|1\rangle$.

Secondly, we dicuss the CNOT gate. The CNOT gate maps

$$\text{CNOT} : |x_1, x_2\rangle \mapsto |x_1, x_1 \oplus x_2\rangle$$

and therefore does not introduce any phases at all. It does not even mix states in a strict sense, it rather *permutes* them.

Using these observations for $-[\phi]_{\mathsf{Z}}-$ and CNOT, we see that it is indeed reasonable to analyze the action of the network by considering the computational basis states. If we insert a basis state into the network, it might collect some phase shift from the z rotations, and it might get permuted with some other basis state by the CNOT gates, but it will not get mixed with any other basis state. Therefore, if we start with a basis state, the system will be in a basis state at any point throughout the network. Still, it might have collected some phase.

Analysis of the network

Now we analyze the network step by step. We start with a basis state as the initial state,

$$|\text{init}\rangle = |x_1, x_2\rangle .$$

The first operation is

$$\mathsf{A} = \begin{array}{l} x_1 -[\phi_{10}]_{\mathsf{Z}}- \\ x_2 \text{———} \end{array} .$$

It maps

$$|x_1, x_2\rangle \equiv |x_1\rangle|x_2\rangle \overset{\mathsf{A}}{\longmapsto} \left(\exp\left[-i\tfrac{\phi_{10}}{2}(-1)^{x_1}\right]|x_1\rangle\right)|x_2\rangle \equiv \exp\left[-i\tfrac{\phi_{10}}{2}(-1)^{x_1}\right]|x_1, x_2\rangle$$

and thus the state $|\mathsf{A}\rangle$ after the application of A is

$$\begin{aligned} |\mathsf{A}\rangle &= \mathsf{A}|\text{init}\rangle \\ &= \exp\left[-i\tfrac{\phi_{10}}{2}(-1)^{x_1}\right]|x_1, x_2\rangle \ . \end{aligned}$$

The operation

$$\mathsf{B} = \quad \begin{array}{l} x_1 \text{———} \\ x_2 \text{—}[\phi_{01}]_{\mathsf{Z}}\text{—} \end{array}$$

accordingly maps

$$|x_1, x_2\rangle \overset{\mathsf{B}}{\longmapsto} \exp\left[-i\tfrac{\phi_{01}}{2}(-1)^{x_2}\right]|x_1, x_2\rangle \ .$$

If applied to $|\mathsf{A}\rangle$, the phase already contained in $|\mathsf{A}\rangle$ has to be added, and we obtain

$$\begin{aligned} |\mathsf{B}\rangle &= \mathsf{B}|\mathsf{A}\rangle \\ &= \exp\left[-i\left(\tfrac{\phi_{10}}{2}(-1)^{x_1} + \tfrac{\phi_{01}}{2}(-1)^{x_2}\right)\right]|x_1, x_2\rangle \ . \end{aligned}$$

Next, the first CNOT operation is applied.

$$\mathsf{C} = \quad \begin{array}{l} x_1 \text{—}\bullet\text{—} \\ x_2 \text{—}\oplus\text{—} \end{array} \quad .$$

CNOT maps

$$|x_1, x_2\rangle \overset{\mathsf{C}}{\longmapsto} |x_1, x_1 \oplus x_2\rangle \ ,$$

i.e., the state $|\mathsf{B}\rangle$ is mapped to

$$\begin{aligned} |\mathsf{C}\rangle &= \mathsf{C}|\mathsf{B}\rangle \\ &= \exp\left[-i\left(\tfrac{\phi_{10}}{2}(-1)^{x_1} + \tfrac{\phi_{01}}{2}(-1)^{x_2}\right)\right]|x_1, x_1 \oplus x_2\rangle \ . \end{aligned}$$

Now the third z rotation follows.

$$\mathsf{D} = \quad \begin{array}{l} x_1 \text{———} \\ x_2 \text{—}[\phi_{11}]_{\mathsf{Z}}\text{—} \end{array} \quad .$$

As mentioned before, it adds a phase of $\exp\left[-i\frac{\phi_{11}}{2}(-1)^{x_2}\right]$ to the state $|x_1, x_2\rangle$. But D is applied to the state $|\mathsf{C}\rangle$, where the register is in the $|x_1, x_1 \oplus x_2\rangle$ state. Consequently, the phase shift in this case is $\exp\left[-i\frac{\phi_{11}}{2}(-1)^{x_1 \oplus x_2}\right]$, i.e.,

$$\begin{aligned} |\mathsf{D}\rangle &= \mathsf{D}|\mathsf{C}\rangle \\ &= \exp\left[-i\left(\tfrac{\phi_{10}}{2}(-1)^{x_1} + \tfrac{\phi_{01}}{2}(-1)^{x_2} + \tfrac{\phi_{11}}{2}(-1)^{x_1 \oplus x_2}\right)\right]|x_1, x_1 \oplus x_2\rangle \ . \end{aligned}$$

Finally, the second CNOT $\equiv \mathsf{E}$ acts on the basis state $|x_1, x_1 \oplus x_2\rangle$, yielding

$$|x_1, x_1 \oplus x_2\rangle \overset{\mathsf{E}}{\longmapsto} |x_1, x_1 \oplus x_2 \oplus x_1\rangle \equiv |x_1, x_2\rangle \ .$$

Therefore, the final state is

$$\begin{aligned} |\text{final}\rangle &= |\mathsf{E}\rangle = \mathsf{E}|\mathsf{D}\rangle \\ &= \exp\left[-i\left(\tfrac{\phi_{10}}{2}(-1)^{x_1} + \tfrac{\phi_{01}}{2}(-1)^{x_2} + \tfrac{\phi_{11}}{2}(-1)^{x_1 \oplus x_2}\right)\right] |x_1, x_2\rangle \ . \end{aligned} \tag{5.8}$$

One could argue that this phase is a global phase, and therefore irrelevant. This is true, of course, as long as we only insert basis states into the network. As soon as we apply the network to some superposition of basis states,

$$|\text{init}\rangle = \sum_{x_1,x_2} \alpha_{x_1,x_2} |x_1, x_2\rangle \ ,$$

we see that the resulting state is

$$|\text{final}\rangle = \sum_{x_1,x_2} \exp\left[-i\left(\tfrac{\phi_{10}}{2}(-1)^{x_1} + \tfrac{\phi_{01}}{2}(-1)^{x_2} + \tfrac{\phi_{11}}{2}(-1)^{x_1 \oplus x_2}\right)\right] \alpha_{x_1,x_2} |x_1, x_2\rangle \ .$$

In this case, the phase factors observed in (5.8) (to be correct, their differences) are *relative* phases—at least, we cannot consider them as global phases which are not important.

Reformulation of the result

The result obtained in (5.8) can be formulated as follows. If applied to an input state $|x_1, x_2\rangle$ the network (5.6) preserves this input state, but a phase θ_{x_1,x_2} is added:

$$|\mathbf{x}\rangle \equiv |x_1, x_2\rangle \mapsto e^{-i\theta_{\mathbf{x}}} |\mathbf{x}\rangle \ ,$$

where the phase angle is

$$\theta_{\mathbf{x}} \equiv \theta_{x_1,x_2} = \frac{\phi_{10}}{2}(-1)^{x_1} + \frac{\phi_{01}}{2}(-1)^{x_2} + \frac{\phi_{11}}{2}(-1)^{x_1 \oplus x_2} \ . \tag{5.9}$$

In order to restate our result more canonically, we first introduce a new phase ϕ_{00}. Clearly, to fit into the pattern of (5.9), ϕ_{00} has to appear in the sum as $\frac{\phi_{00}}{2}(-1)^0$, since 0 is the empty XOR expression. Therefore, including ϕ_{00} in (5.9) gives

$$\theta_{\mathbf{x}} \equiv \theta_{x_1,x_2} = \frac{\phi_{00}}{2} + \frac{\phi_{10}}{2}(-1)^{x_1} + \frac{\phi_{01}}{2}(-1)^{x_2} + \frac{\phi_{11}}{2}(-1)^{x_1 \oplus x_2} \ . \tag{5.10}$$

Clearly, this phase $\phi_{00}/2$ *is* a global phase—it is applied the same way to *all* the basis states. Therefore, we can safely include it in $\theta_{\mathbf{x}}$, since it still describes the physical behavior of the network (5.6).

For the following, we define vectors $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ as

$$\begin{aligned} \boldsymbol{\theta} &:= (\theta_{00}, \theta_{01}, \theta_{10}, \theta_{11}) \quad \text{and} \\ \boldsymbol{\phi} &:= (\phi_{00}, \phi_{01}, \phi_{10}, \phi_{11}) \ . \end{aligned}$$

We also give a name to our network (5.6), namely $U_{\boldsymbol{\theta}}$, the oracle belonging to $\boldsymbol{\theta} \equiv \boldsymbol{\theta}(\boldsymbol{\phi})$. It does

$$U_{\boldsymbol{\theta}} : |\mathbf{x}\rangle \mapsto e^{-i\theta_{\mathbf{x}}(\boldsymbol{\phi})}|\mathbf{x}\rangle \ .$$

On the other hand, recall that our goal is to build the different possible Deutsch oracles, which actually correspond to

$$U_f : |\mathbf{x}\rangle \mapsto (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle \ .$$

So in order to implement a Deutsch oracle using the network $U_{\boldsymbol{\theta}}$, i.e., $U_f = U_{\boldsymbol{\theta}}$, we require

$$e^{-i\theta_{\mathbf{x}}(\boldsymbol{\phi})} = (-1)^{f(\mathbf{x})} \ , \tag{5.11}$$

which finally leads to the requirement[4] that

$$\theta_{\mathbf{x}}(\boldsymbol{\phi}) = \pi f(\mathbf{x}) \ , \tag{5.12}$$

or by defining $\boldsymbol{f} = (f(00), f(01), f(10), f(11))$,

$$\boldsymbol{\theta}(\boldsymbol{\phi}) = \pi \boldsymbol{f} \ .$$

The question which remains open is whether $\boldsymbol{\phi}$ —recall that this is the adjustable variable in our network $U_{\boldsymbol{\theta}}$ —can be chosen such that we get the appropriate $\boldsymbol{\theta}$. In order to find out about this we rewrite the relation (5.10) between $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, which is a linear mapping, as a matrix:

$$\boldsymbol{\theta} = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \boldsymbol{\phi} \ . \tag{5.13}$$

But this matrix is the (unnormalized) Hadamard transformation H_2 in two dimensions, and we know very well that this matrix can be inverted. In fact, it is its own inverse (up to the normalization factor):

$$H_2 H_2 = 2^2 = 4 \ ,$$

so that

$$\boldsymbol{\phi} = \frac{1}{2} H_2 \boldsymbol{\theta} \ . \tag{5.14}$$

This implies that for any desired (balanced) function f, we can first derive $\boldsymbol{\theta}$ using (5.12). By virtue of (5.14) we can then determine $\boldsymbol{\phi}$ such that the mapping $U_{\boldsymbol{\theta}} \equiv U_{\boldsymbol{\theta}(\boldsymbol{\phi})}$ gives exactly the oracle U_f for f. Therefore, the network $U_{\boldsymbol{\theta}}$ of (5.6) allows us to construct any oracle with the same setup, simply by changing the three angles ϕ_{x_1,x_2} in the network. In this sense it is a programmable network, since it

[4] In fact, (5.12) is only one possible choice of $\boldsymbol{\theta}$ satisfying (5.11).

can be programmed to implement any Boolean function simply by adjusting the z rotations. (Admittedly, until now this only works for two qubits, which is not spectacular at all, but this was only to demonstrate how these networks work *in principle*. We will show how to extend this to an arbitrary number of qubits in a moment.)

We should add that this network is in fact capable of implementing $U_{\boldsymbol{\theta}} : |\mathbf{x}\rangle \mapsto e^{-i\theta_{\mathbf{x}}}|\mathbf{x}\rangle$ for *any* $\boldsymbol{\theta}$ with the same fixed setup, only by varying the rotation angles of the z rotations.

Generalization to N qubits

In order to generalize this to an arbitrary number of qubits—which gives us a universal and powerful method to build arbitrary Deutsch oracles—we start in the reverse direction. We look for an N-qubit network with parameters $\boldsymbol{\phi} \in [0, 2\pi]^{2^N}$ (rotation angles, e.g.), which does

$$U_{\boldsymbol{\theta}} : |\mathbf{x}\rangle \mapsto e^{-i\theta_{\mathbf{x}}}|\mathbf{x}\rangle \ , \tag{5.15}$$

and where $\boldsymbol{\theta}$ is related to $\boldsymbol{\phi}$ by

$$\boldsymbol{\theta} = \frac{1}{2} H_N \boldsymbol{\phi} \ , \tag{5.16}$$

with H_N the unnormalized N-bit Hadamard transform. This leads to the inverse relation

$$\boldsymbol{\phi} = \frac{1}{2^{N-1}} H_N \boldsymbol{\theta} \ . \tag{5.17}$$

By the same arguments as in the two-bit case, this would give us the possibility to easily adjust the network (i.e., the ϕs), such that it implements the desired Deutsch oracle—the network would be programmable.

Such a network does not mix the computational basis states $|\mathbf{x}\rangle \in \{0,1\}^{2^N}$, but the phase of each of these states is shifted by $\theta_{\mathbf{x}}$. Applying the definition (2.6) of the Hadamard transform on N qubits to (5.16), we obtain

$$\begin{aligned} \theta_{\mathbf{x}} &= \frac{1}{2} \sum_{\mathbf{y}} (-1)^{\mathbf{x}\cdot\mathbf{y}} \phi_{\mathbf{y}} \\ &= \sum_{\mathbf{y}} (-1)^{x_1 y_1 \oplus \ldots \oplus x_N y_N} \frac{\phi_{\mathbf{y}}}{2} \ . \end{aligned}$$

This means that—if applied to any basis state $|\mathbf{x}\rangle$ —our network has to shift the phase by $(-1)^{x_1 y_1 \oplus \ldots \oplus x_N y_N} \phi_{\mathbf{y}}/2$, for all values of $\mathbf{y}$.

In this sum, we can safely omit the contribution from $\mathbf{y} = \mathbf{0}$, since in this case the contribution to the phase of a state $|\mathbf{x}\rangle$ is $(-1)^0 \phi_{\mathbf{0}}/2 = \phi_{\mathbf{0}}/2$, i.e., a

global phase.[5] Therefore, we have to realize a network which applies the array of operations

$$|\mathbf{x}\rangle \mapsto \exp\left[-i\tfrac{\phi_{\mathbf{y}}}{2}(-1)^{\mathbf{x}\cdot\mathbf{y}}\right]|\mathbf{x}\rangle \tag{5.18}$$

to each state, where the array is taken over all $\mathbf{y} \neq \mathbf{0}$. For these $\mathbf{y}$, one can easily see that for $\mathbf{y}$ fixed $\mathbf{x}\cdot\mathbf{y} = x_1y_1 \oplus \ldots \oplus x_Ny_N$ is exactly the $\oplus$-sum of all the x_i for which $y_i = 1$ holds, i.e., $\bigoplus_{\{i|y_i=1\}} x_i$. This is true since all the products with $y_i = 0$ vanish and therefore do not contribute.

Now for a given $\mathbf{y} \neq \mathbf{0}$, the operation (5.18) can be accomplished the following way. Start by generating the desired $\oplus$ (i.e., XOR) combination of the input bits. This can be done by choosing one of the bits as the target bit, and applying CNOT between all other bits and the target bit. Thereby, the desired XOR combination $\mathbf{x}\cdot\mathbf{y}$ of the input bits is generated on the target bit. Then perform a z rotation by an angle $\phi_{\mathbf{y}}$, and then apply the CNOT network backwards to restore the original state in the register (since CNOT $\cdot$ CNOT $= I$). This network obviously does not change basis states, but it shifts their phases according to (5.18).

The whole programmable network for (5.15) can be obtained by performing (5.18) as described above for *all* $\mathbf{y} \neq \mathbf{0}$ one after another. In principle, this yields already a programmable network for an arbitrary number of qubits which obeys the equations mentioned above.

However, this method can be simplified considerably, even at the current conceptual level. Particularly, since the effect of a z rotation is determined solely by the the state of the qubit the rotation is applied to, the states of the other qubits are not important. Neglecting the z rotations for a moment, we only have to find a CNOT network which generates all possible XOR combinations of the inputs and finally restores the original state. This network can be regarded as a classical logical network in the same sense in which CNOT can be regarded as a classical logic gate. For this reason, we might as well use the CNS gate introduced in Chapter 4 as the basic building block for these networks. Into that classically derived network, the z rotations have to be placed on the qubits which currently have the right XOR value of the input bits.

By the procedure given above, it is clear that for any qubit number a network of this kind can be implemented in principle. We thus know how to build a network which is capable of doing any diagonal phase-shifting operation

$$U_{\boldsymbol{\theta}} : |\mathbf{x}\rangle \mapsto e^{-i\theta_{\mathbf{x}}}|\mathbf{x}\rangle \ .$$

The desired phase shift $\boldsymbol{\theta}$ can be adjusted arbitrarily, since the rotation angles of

[5] Note that—since we are tracing backwards— omitting $\phi_{\mathbf{0}}/2$ corresponds to adding $\phi_{00}/2$ in the two-bit example. In fact the addition of some constant $\phi_{00}/2$ would not have been necessary at all (since it has no physical consequences!) to make the method work. But we would not have obtained the Hadamard transform as the relation between $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, thus making the treatment less handy.

the z rotations can be determined easily using

$$\boldsymbol{\phi} = \frac{1}{2^{N-1}} H_N \boldsymbol{\theta} \; . \tag{5.19}$$

For the unnormalized Hadamard transform, efficient classical algorithms are known [Knu97] which particularly are more efficient than a simple matrix multiplication. Namely, one only needs $N2^N$ additions (although H_N is a $2^N \times 2^N$ matrix).

Specifically, this kind of network network allows for the implementation of

$$U_f : |\mathbf{x}\rangle \mapsto (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle$$

for any Boolean function f, especially for any balanced function and thus for any Deutsch oracle. The network is the same for all oracles considered; the only parameters which have to be tuned are the rotation angles of the considered $(2^N - 1)$ z rotations. The rotation angle can be determined easily using $\boldsymbol{\theta} = \pi \boldsymbol{f}$ in addition to (5.19). Therefore, we call this network a *programmable network* since it can be programmed easily to implement various functions or functionalities.

5.4.2 Implementing the programmable networks

In the following we give some methods how to construct the desired networks. Each of these methods is easier to apply and leads to more compact sequences than the theoretical construction used in the existence proof above. Hopefully, the examples given in the following also help to clarify the underlying idea of the programmable networks.

Gray sequences

We start with a method how to get efficient networks for systems providing direct CNOT operations between all pairs of qubits. This method is based on a suggestion how to construct similar networks for N-bit Toffoli gates given in [BBC$^+$95]. In fact, these networks gave the first idea to the programmable networks. We will just briefly present the method by showing how the construction works for three and four qubits. We will not discuss the fully formalized version of this network, since the formalization is straightforward and we actually wanted to focus on systems which only provide nearest-neighbor coupling.

For this construction, we first have to introduce Gray sequences. An N-bit Gray sequence is a sequence of all 2^N N-bit binary numbers, where two adjacent numbers differ by exactly one bit. (Optimally, this also holds for the first and the last number of the sequence.) There exist many different Gray sequences for each N (see [Knu02] for an exhaustive survey), but for our purpose, the simplest construction will suffice.

We denote the N-bit Gray sequence by $G_N = (g_1^N, \ldots, g_{2^N}^N)$. The simplest Gray sequence is the one-bit sequence $G_1 = (0, 1)$. If G_N is known, G_{N+1} can be constructed easily as $G_{N+1} = (0G_N, 1\overline{G_N})$, where $\overline{G_N} = (g_{2^N}^N, \ldots, g_1^N)$ is the reversed Gray sequence for N bits, and $0G_N$ denotes the sequence consisting of G_N, with the $N+1$st bit filled with 0. One can easily check that this construction ensures that G_{N+1} is a Gray sequence, provided G_N is one. The sequences for up to four qubits are given in Table 5.3.

Now in order to find the desired network simply generate the XOR combinations of the x_is in the order in which they appear in the Gray sequence, where an "1" on the ith position symbolizes that the corresponding x_i appears in the XOR combination (as opposed to the usual convention, we number the x_is from right to left to render the resulting networks comparable). Clearly, $0 \cdots 0$ has to be omitted. The target bit for the CNOTs is the one which stays "1" in the Gray sequence for the longest time. In Table 5.3, we marked the target bits for the CNOTs by bold typesetting. In order to illustrate the method, we give the corresponding CNOT networks for two, three and four qubits in Fig. 5.2.

This method gives the most efficient implementation of such networks for systems with CNOT between all qubits, unless parallel execution is considered. The number of CNOT gates required is $2^N - 2$. Since we have to generate $2^N - (N + 1)$ XOR combinations and finally to restore the original state of the register, this is indeed optimal. In principle this construction could be transferred to systems with only nearest neighbor coupling. Still, this would require for the swapping of qubits. We need roughly N swappings per CNOT, namely $N/2$ before and after each CNOT. Therefore, the number of CNOT gates grows by a factor of N. Of course, one could argue that some of the SWAP gates probably cancel out, and that the CNS gate could be used to save SWAPs. Although this is true in principle, such a workaround is not desirable since we loose the clarity of the solution and have to start with tedious simplifications again.

Construction for Josephson qubits

In the following, we give another method for constucting non-parallel networks. The method only presumes a chain of qubits with nearest neighbor coupling, and is suitable for all types of interaction, although the best results are achieved for the CNS gate. It is therefore the optimal construction for our hardware setup of Josephson junctions coupled by SQUID loops. It is especially easy to use, since the network for $N + 1$ qubits is obtained inductively by applying a simple substitution rule to the N-qubit network.

We start by reconsidering the requirements for such a network. The N-qubit network has to contain all possible XOR combinations of the input bits $(x_1, \ldots, x_N)$, and perform z rotations on them. The $(N + 1)$-qubit network, on the other hand, has to generate all XOR combinations of $(x_1, \ldots, x_{N+1})$. Now these XOR combinations of the $N + 1$ bits of course contain all the XOR combina-

G_1	G_2	G_3	G_4
0	0|0	0|00	0|000
1	0|1	0|01	0|001
	1|1	0|**1**1	0|0**1**1
	1|0	0|**1**0	0|0**1**0
		1|10	0|**1**10
		1|11	0|**1**11
		1|01	0|**1**01
		1|00	0|**1**00
			1|100
			1|101
			1|111
			1|110
			1|010
			1|011
			1|001
			1|000

Table 5.3: Gray sequence for up to four bits, constructed by the method explained in the text. Except from the new bit, the new sequence contains the old sequence in its original order, and afterwards in the reversed order. The bold "**1**"s denote the target bits for the CNOTs.

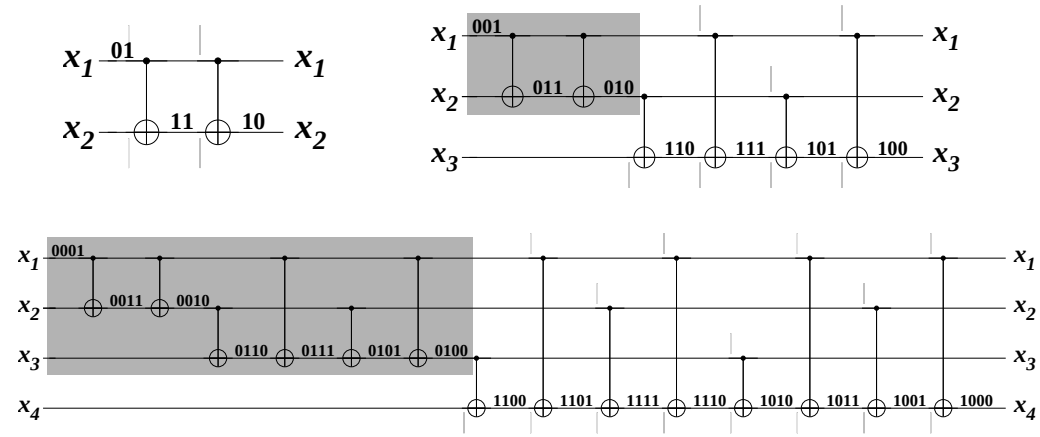


Figure 5.2: Programmable networks (z rotations omitted) for two, three and four qubits, constructed using Gray sequences. The blocks marked gray are identical to the programmable network for one qubit less.

tions of the first N qubits $(x_1, \ldots, x_N)$, as well as these combinations XORed with x_{N+1}. As one can easily see (or check by counting the number of different XOR combinations), together with the "XOR combination" x_{N+1} itself, this already gives *all* XOR combinations of $(x_1, \ldots, x_{N+1})$.[6]

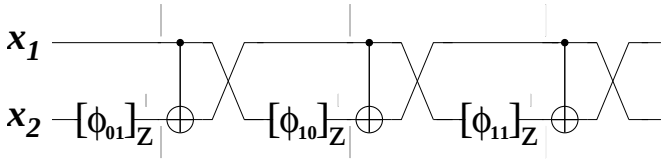
We therefore see that—starting from the N-qubit network—we get the $N+1$-qubit network by replacing each z rotation by the z rotation for the old condition, plus a z rotation on the same qubit XORed with the $N+1$st qubit. Finally, we also have to add a z rotation for the new qubit x_{N+1} alone. In order to be able to realize the additional XOR with the $N+1$st qubit in a system with nearest neighbor coupling, we require all the z rotations to be located on the Nth qubit, i.e., next to the newly added qubit. This enables us to perform the XOR, i.e., CNOT, operation of the line holding the old XOR combination with the new qubit directly.

In the following, we illustrate the method by demonstrating the construction of the network for up to three qubits, starting with the trivial one-qubit case. There, the network consists of a single z rotation:

(5.20)

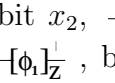
For the two-qubit case, we could in principle use the network (5.6). Nevertheless, since we want to have *all* the z rotations located on the second line, a bit more effort has to be done.

A network which gives all the z rotations on the second line using the CNS gate as the basic two-qubit gate might look like

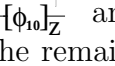


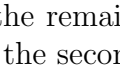
(5.21)

The three CNS gates cancel out, and the network provides all necessary XORs.

This case already provides all ingredients needed to derive a general rule. Namely, the step from the one-qubit network (5.20) to the two-qubit network (5.21) can be described as follows. Firstly, we added the phase shift controlled by the new qubit x_2, 

. Then, we replaced the old phase shift with the condition x_1,

, by two phase shifts with the two XOR conditions x_1 and $x_1 \oplus x_2$, i.e., 

and 

. This was accomplished by replacing the old z rotation by the remaining part of the network. Note that all the phase shifts are located on the second qubit.

[6] Speaking more formally, we split the sum over all $(y_1, \ldots, y_{N+1})$, where the XORs are $(x_1, \ldots, x_{N+1}) \cdot (y_1, \ldots, y_{N+1})$, into a sum over all $(y_1, \ldots, y_N, 0)$ and all $(y_1, \ldots, y_N, 1)$ which gives the N-qubit XORs, where in the second case x_{N+1} is additionally XORed. The special treatment of x_{N+1}, i.e., $(0, \ldots, 0, 1)$, results from the fact that the empty XOR combination on N qubits is missing. If we add a formal phase shift for $\mathbf{y} = 0$, the extension gets fully canonical.

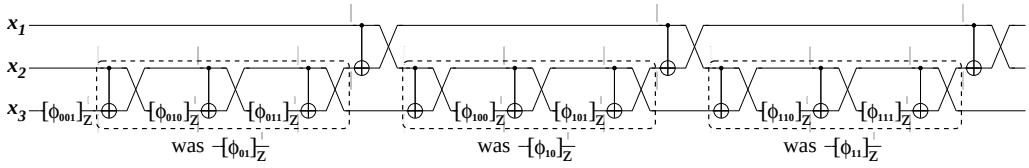
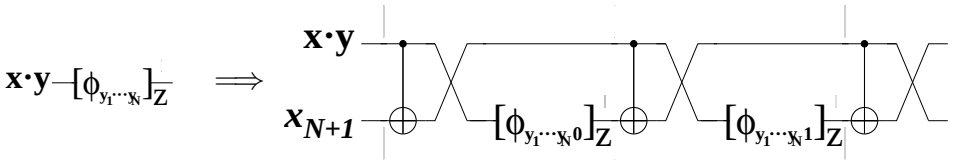


Figure 5.3: Programmable network for three qubits, using CNS and nearest neighbor coupling, derived by the recursive method given in Section 5.4.2. The blocks obtained by substituting (5.22) into the two-bit network (5.21) are marked, and the original shifts located on these blocks in the two-bit network are given.

In fact, this is already the step from N to $N+1$ we discussed above. Provided we have the network for N qubits, we can obtain the network for $N+1$ qubits the following way. Start by adding the phase shift acting only on the new qubit x_{N+1}, i.e., $-[\phi_{0\cdots01}]_Z-$, and then simply replace all conditional rotations $-[\phi_{y_1\cdots y_N}]_Z-$ of the old network according to the following rule:



. (5.22)

Following these rules, we can derive the three-bit network from the two-bit network, and generally, $N+1$ from N. Especially, this construction leaves all the z rotations on the last qubit, so that the construction can be iterated. Additionally, the indices $\mathbf{y} = (y_1, \ldots, y_{N+1})$ of the rotations, i.e., the XOR conditions, appear in their natural order if read as a binary code.

We give the network for three qubits as derived from the two-qubit network in Fig. 5.3, where the blocks replaced relative to the two-bit network (5.21) are highlighted. We omit the four-qubit network, since the examples get lengthy very quickly. The number of CNS gates needed is $3(2^N - (N+1))$.

We close this subsection with a remark concerning other interaction types. If the genuine interaction of the system is the ZZ rather than the XY interaction, i.e., we have CNOT instead of CNS as the basic two-qubit operation, the network can be built by noting that two antiparallel CNOT gates exactly give a CNS gate. As far as we can see, this is the best possible realization of the extension rule (5.22) with CNOT.[7] For the JJ interaction, things are more complicated. Probably the most efficient method is to use one CNOT, one CNS and one SWAP operation in the extension rule (5.22) where the gates are built using the JJ interaction.

Parallel construction

In the following we present a different method for the construction of the programmable networks which gives by far more compact sequences than the ap-

[7]There is another possibility, but the total number of CNOTs in (5.22) is still 6.

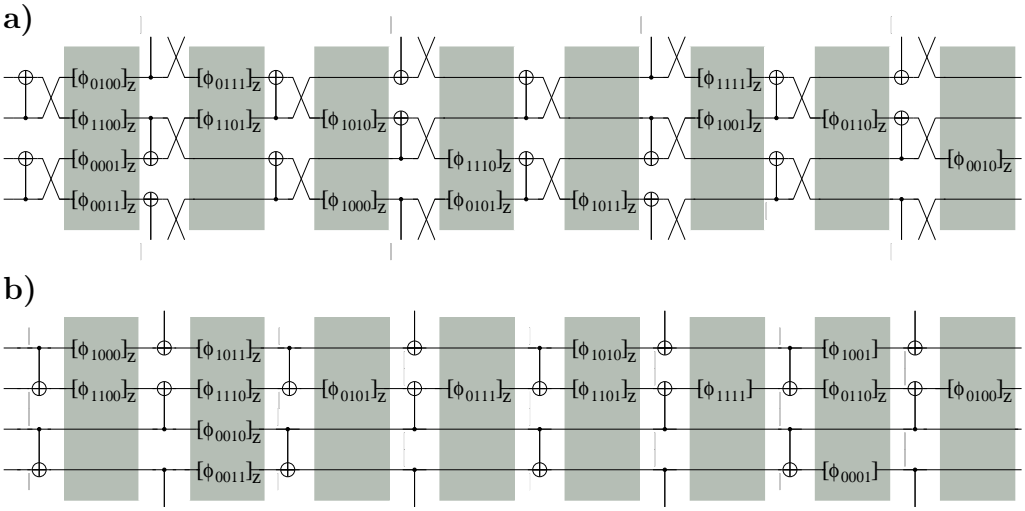


Figure 5.4: a) The four-bit network for a system with nearest neighbor XY coupling, where parallel execution of operations is feasible. In this setup, the qubits are arranged in a ring. Note that the z rotations and the two-qubit operations are located in distinct blocks. Compare the length of this sequence to to the serial *three*-bit network of Fig. 5.3. **b)** The corresponding circuit with the CNOT gate as the basic operation. Obviously, in this case the use of the CNS gate does not make the sequence shorter. It is interesting to note the regularity of both networks. The regularity of a) can even be increased by rotating the origin with the network.

proaches mentioned before, and still works for nearest neighbor coupling. The difference to the other approaches is the parallel execution of gates which—compared to serial execution—can give a speed-up of up to a factor of $O(N)$. The main drawback is that we cannot give canonical ways how to construct these networks efficiently. We put the discussion of this point off until the end of the section.

As already mentioned at the end of Section 5.4.1, pg. 79, the only thing that matters about the network is that it only consists of operations as CNOT and CNS which perform permutations on the basis, but do not change the phase, and z rotations. These phase shifting rotations have to be placed on qubits having exactly the desired XOR combination of the input bits—the state of all the other qubits is of no importance. This allows for the parallel execution of operations as long as these boundary conditions are respected. The parallelization itself imposes no problems since operations acting on different qubits one after another commute and therefore can be executed in parallel as well.

The main problem with the parallel networks is that we could not find a method for the formal construction of these networks. The network shown in Fig. 5.4a shows one optimal (it is not necessarily unique) universal four-bit network for a circular setup of qubits with nearest-neighbor coupling, found by an

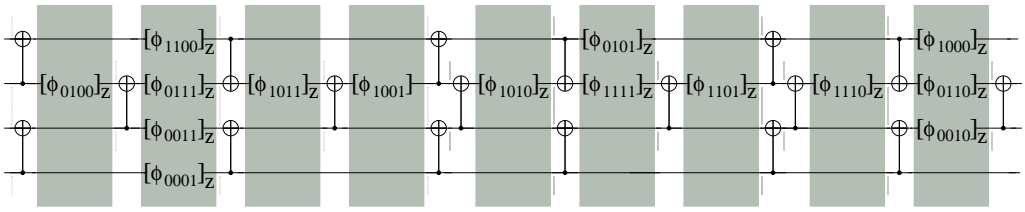


Figure 5.5: The four-bit programmable network for a setup of qubits arranged in a line with only nearest neighbor coupling. Once again, note the high regularity of the sequence. A sequence of the same length can also be obtained using CNS as the basic building block.

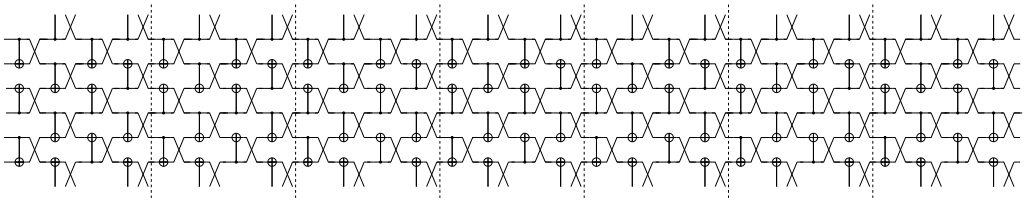


Figure 5.6: The six-qubit programmable network for a setup of qubits arranged in a ring, with CNS between nearest neighbors. The z rotations have been omitted, since they would not have been readable anyway. The network was found by an algorithm calculating blocks containing 4 steps each, and then repeatedly applying that block. The optimization was only performed within that block.

4 bits				**6 bits**	
lower bound	Fig. 5.4a	Fig. 5.4b	Fig. 5.5	lower bound	Fig. 5.6
8	8	8	12	21.3	28

Table 5.4: Length (i.e., number of blocks of two-bit operations) of the parallel sequences given in Figs. 5.4–5.6, compared to the theoretical lower bound. Note that this lower bound does not take into account the restricted interaction range and that the six-bit sequence of Fig. 5.6 is not necessarily the shortest possible sequence.

exhaustive search. Note that this search is purely combinatoric and does not have to take care of the quantum mechanical nature of the network. Interestingly, the sequence of CNS operations is highly regular, suggesting the existence of some rule behind the network. The network is well structured—there are blocks containing only CNS gates (note the massive parallelism!) and there are blocks containing only phase shifters. Due to the structure of the network the position of most phase shifts is not unique. Note also that the sequence consists of exactly $16 \equiv 2^N$ two-bit operations, only two more than for the (not nearest neighbor) network with Gray sequences. In fact, even a sequence with only 14 CNS gates does exists, which is optimal as already explained for the Gray sequence technique. Still, we did not present that sequence here since it appears less regular—two of the CNS operations are carried out in a non-parallel way.

Studying the four-bit network of Fig. 5.4a, one might be lead to the conclusion (possibly motivated by Chapter 4) that the use of the CNS gate is essential for the compactness of the code. Due to the arrangement of the qubits in a ring the SWAP included in the CNS operation somehow rotates two of the qubits clock- and the other two counterclockwise, thus mixing them well. It turns out, though, that this argument is not completely correct. Namely, we give the same network with CNOT gates in Fig. 5.4b, which obviously has exactly the same length. Still, there is some advantage of the CNS gate over the CNOT gate in case one uses the programmable networks for a fixed purpose. An example is given in Section 5.5.3.

Of course, one can also search for networks in a linear setup. Also in this case, the CNS gate and the CNOT gate yield sequences of the same length. The result for CNOT is given in Fig. 5.5.

Finally, we also give a network for six qubits (the five-qubit case is less interesting, since it does not allow for more parallelism than the four-qubit case), using CNS gates and qubits arranged in a ring (Fig. 5.6). This network was not derived by an full exhaustive search but—motivated by the "periodicity" of the networks in Fig. 5.4—by repeatedly applying identical blocks of length 4 and only searching within all the possibilities for that block.

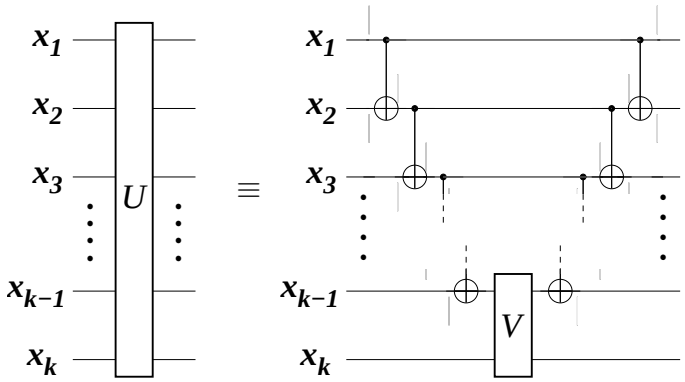
The minimum length of the sequences with a parallel implementation is $2^N/(N/2)$. Namely, (roughly) 2^N XOR combinations have to be generated. Per time step, at most $N/2$ combinations can be generated, because both CNOT and CNS return one new and one of the old states. In Table 5.4, we compare this bound to the length of the sequences given in this section.

5.4.3 Similar approaches

At the end of this section, we want to discuss two results obtained by other groups which are related to the approach presented here. In fact, our solution can be regarded as a combination of these two approaches although it was not found that way.

The first paper we want to mention deals with the implementation of arbitrary unitary operations using only the Hamiltonian allowed by spin-1/2 systems [KLL00]. The authors start by noting that the generator G of any unitary U, $U = e^{-iG}$, can be decomposed with respect to a basis $\{B_s\}$ which consists of all possible outer products of the Pauli spin matrices and the identity matrix: $U = e^{-i\sum_s b_s B_s}$. The key step is now to transform this representation into another one, namely $U = \prod_s e^{-ib'_s B_s}$. The authors emphasize that it is not clear at all how this transform might work in general.

Finally—and this is the step we are most interested in—they show how the $e^{-ib'_s B_s}$ can be realized in a system with only ZZ coupling between nearest neighbors. To this end, they first note that any $e^{-ib'_s B_s}$ can be transformed into some $e^{-id_s D_s}$ by local unitaries where D_s is an outer product of only σ_z and identity matrices. The implementation of $e^{-id_s D_s}$ still would require ZZ type interaction between *multiple* qubits, i.e., Hamiltonians of the form $\sigma_z^{(i_1)} \otimes \cdots \otimes \sigma_z^{(i_k)}$. But such Hamiltonians can be simulated by the following circuit:



(5.23)

where $U = e^{-id_s \sigma_z \otimes \cdots \otimes \sigma_z}$, and $V = e^{-id_s \sigma_z \otimes \sigma_z}$. This can be understood nicely in terms of our XOR formalism. U is a diagonal matrix doing $U : |x_1, \ldots, x_k\rangle \mapsto \exp\left[-id_s(-1)^{(x_1 \oplus \cdots \oplus x_k)}\right] |x_1, \ldots, x_k\rangle$. But we know how to implement such matrices! Just create an XOR of the corresponding bits, apply a z rotation about an angle of $2d_s$ to the XORed qubit, and undo the XORs. That is indeed what is happening in (5.23), except that the last pair of CNOTs and the phase shift are collected to the operation V, which can be checked using the XOR network for two bits.

Secondly, we refer to a preprint named *Concurrent Quantum Computation* [YMY00]. The authors start from the point that mappings of the form $U : |\mathbf{x}\rangle \mapsto (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle$ are needed often in quantum computation and that the construction of these operations by sequential application of one- and two-bit operations takes a vast number of operations. They note that on a system providing arbitrary and independently tunable multi-particle ZZ interactions (!) one can implement U with only one time evolution. They show that this can be done by properly adjusting the interactions strengths resp. interaction times of the different interactions.

But as we have seen from the other paper, each multi-particle ZZ interaction can be implemented by creating XORs of the corresponding bits and applying a z rotation with a properly chosen angle. Therefore, the combination of both approaches plus the observation that the relation between function and phase shifts, i.e., interaction times, is exactly the Hadamard transform gives our result. Moreover, in our approach it is clear that the key point is *only* the creation of the XOR combinations and the phase shifts, and by no means their order, parallelism, the complete deconstruction of each XOR each time, and so on.

5.5 Other applications for programmable networks

Before closing the work with some considerations on complexity, we want to show briefly some other fields of application for the universal networks introduced above.

5.5.1 Grover's algorithm

Except for Deutsch's algorithm, there is another application of the universal networks in the field of quantum algorithms—the oracle in Grover's algorithm. Recall that Grover's algorithm searches an unsorted database for marked items. The database is represented by an arbitrary function $f : \{0,1\}^N \to \{0,1\}$, and the implementation of the database on the quantum computer works via $U_f : |\mathbf{x}\rangle \mapsto (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle$. Clearly, we can realize such mappings, so that we are capable of testing Grover's algorithm for an *arbitrary* database, especially with an *arbitrary* number of marked items. In fact, we only need our network and Hadamard transforms on all qubits in between since the algorithm only contains the oracle, the Hadamard transform, and the oracle belonging to the database with only item $\mathbf{0}$ marked.

It should be added that for databases with only a few marked items, this method does not give efficient implementations for high N. Namely, for one marked item it is much more efficient to use N-bit controlled phase flips where the state to be flipped can be determined by properly negating certain bits. Strategies for implementing controlled phase flips in $O(N^2)$ operations (or even $O(N)$ if using ancillae) are known [BBC$^+$95]. It should be added that—for systems with arbitrary couplings—these methods perform better only up from $N = 7$. Moreover, for $N = 4$, e.g., the parallel networks given in Fig. 5.4 provide a much more efficient implementation of the Grover oracle as well as of the $\mathbf{0}$ oracle contained in the algorithm itself. Finally, the use of the programmable networks for Grover's algorithm once again gives *one* universal operation sequence for *all* oracles.

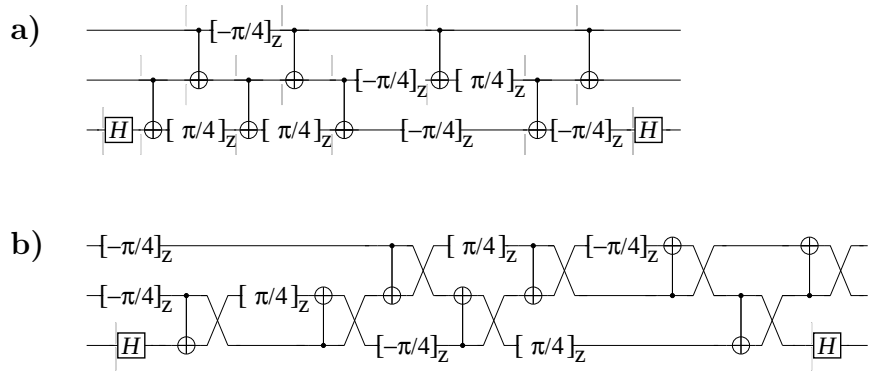


Figure 5.7: Networks for the Toffoli gate using **a)** CNOT and **b)** CNS gates as the basic two-qubit operations. The networks have been derived using the programmable networks, thus giving an efficient implementation.

Up to now, implementations of Grover's algorithm searching for single items have been given for two [CGK98, JMH98, YSV+99, MXKL00, VLS+01, Fen01, EKW01, EF02, YMB+02] and three qubits [VSS+00, KMSW00]. For two qubits, there also exist implementations of the multi-item search [LYL+01, ZLSD02].

5.5.2 The Toffoli gate

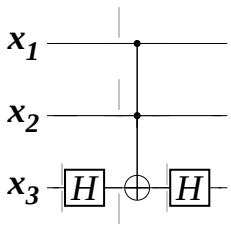
In the following, we turn to more practical applications of the concept of programmable networks. By this, we mean applications which clearly give an advantage over known constructions, and where the operations under consideration could be of practical relevance in future quantum computers.

As a first example, we consider the Toffoli gate. We already gave an efficient implementation of the Toffoli gate in Chapter 4, demonstrating that the use of the CNS gate could lead to considerable more compact sequences for certain operations in case only nearest neighbor couplings are provided. In the following, we show how even shorter sequences for the Toffoli gate in systems with only nearest-neighbor coupling can be obtained, regardless whether the basic gate is CNOT or CNS.

In order to apply the programmable networks to the Toffoli gate, we have to find a connection between the non-diagonal Toffoli gate,

$$\text{TOFFOLI} \equiv \begin{array}{l} x_1 \\ x_2 \\ x_3 \end{array} \equiv \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 0 & 1 \\ & & & & & & 1 & 0 \end{pmatrix},$$

and our diagonal phase-shifting matrices. Obviously, the submatrices with respect to the third qubit, i.e., the 2×2 matrices on the diagonal, are identity and σ_x (i.e., NOT) matrices—and σ_x can be transformed to σ_z by virtue of the Hadamard transform.[8] By Hadamard-transforming the third qubit, we indeed get



$$\equiv \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & -1 \end{pmatrix} .$$

But this is a kind of operation we *can* implement with our universal networks! By Hadamard-transforming the coefficients, we find for the rotation angles

$$\boldsymbol{\phi} = \left(\frac{\pi}{4}, -\frac{\pi}{4}, -\frac{\pi}{4}, \frac{\pi}{4}, -\frac{\pi}{4}, \frac{\pi}{4}, \frac{\pi}{4}, -\frac{\pi}{4}\right) .$$

This can be implemented using a three-bit network. The network can be optimized to the given interaction type and distance by an exhaustive search. For CNOT and coupling between *all* qubits, one finds a construction very similar to the one given in [DiV98][9] (see also pg. 54). In Fig. 5.7, we give the networks for a system with nearest neighbor interaction and with CNOT resp. CNS as the basic two-qubit gate. Observe that the CNS version performs slightly better than the optimized version given in Chapter 4.

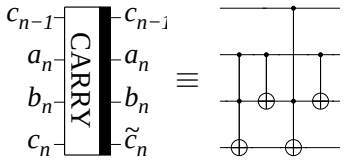
This idea can be extended to a higher number of qubits. The standard construction for N-bit controlled NOTs given in [BBC$^+$95] is the most optimal for up to 7 qubits, and our approach gives us a more general view of this construction. This allows us to optimize the circuit more effectively, especially in systems with restricted interaction types and distances. Moreover, our construction provides easy-to-use methods to parallelize these circuits, which leads, e.g., to a very compact sequence for the four-bit controlled NOT gate. In cases like this it is not a real disadvantage that we have no systematic methods to construct the general parallel N-qubit network since we only need the network for a certain fixed number of qubits, and also the optimization has to be done only once.

[8] This can in fact be expressed more formally. Consider a mapping (as Toffoli) which applies NOT to the Nth qubit, if some function $f(x_1, \ldots, x_{N-1})$ is true, else it applies the identity. By a Hadamard transform on the Nth qubit before and after the operation, σ_x is transformed to σ_z, while the identity does not change. Therefore, one obtains a mapping σ_z iff $f(x_1, \ldots, x_{N-1}) = 1$.

[9] Indeed, these constructions for the TOFFOLI gate, namely the one given in [BBC$^+$95], lead to the idea for the programmable networks.

5.5.3 The CARRY gate

As another practical example where the use of the programmable networks leads to considerably shorted sequences, we discuss the CARRY gate [VBE96]. The CARRY gate is used in circuits performing a reversible addition, which in turn is necessary to implement the modular exponentiation in Shor's factoring algorithm. The CARRY gate takes two (qu)bits to be added, a_n and b_n, and the carry bit of the last stage of the adder, c_{n-1}. It then returns a carry bit c_n which is one exactly if at least two of the three inputs are one—this is clearly what a carry bit should do. Quantum mechanically, we need to insert a qubit for c_n as well. If the carry bit has to be set, this qubit is negated. The CARRY gate can be implemented by the following circuit [VBE96]:



. (5.24)

As described, the CARRY gate only changes the last qubit, namely applies σ_x to it if the CARRY condition is true, otherwise the identity operator.

By using the hint we already introduced for the Toffoli gate, i.e., by applying the Hadamard transform to the last qubit, we can transform this to a σ_z matrix. Thus, the CARRY gate is transformed to a diagonal phase-shifting matrix. The calculation of the phase shifts for our programmable network gives

$$\boldsymbol{\phi} = \left(\frac{\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{4}, \frac{\pi}{4}, -\frac{\pi}{4}, \frac{\pi}{4}, 0, 0, -\frac{\pi}{4}, \frac{\pi}{4}, 0, 0, 0, 0, \frac{\pi}{4}, -\frac{\pi}{4}\right) .$$

These values could of course be inserted directly into the circuits derived in the sections before, plus the Hadamard transforms on the last qubit. But $\boldsymbol{\phi}$ has only nine components (neglecting ϕ_{0000}) which are non-zero—the other phase shifts can be omitted. But if we do not need to perform phase shifts on these XOR combinations of the qubits, then in fact there is no need to generate these XORs at all! Therefore, we can re-optimize our network to generate only those XOR combinations we actually need. In this case, the CNS gate has an advantage over CNOT. Since the sequence gets shorter, the ability of the CNS gate to mix the states quickly once again comes into play. If we consider a linear setup of qubits with nearest neighbor coupling, using CNS we need 10 gates, or 12 gates in 8 slices for a parallel implementation. For CNOT, we need 14 gates resp. 15 gates in 11 slices. Compare this to the $2 \cdot 6 + 2 = 14$ CNOTs needed for the conventional implementation of the sequence given in (5.24) although we assume arbitrary couplings! In a system of qubits arranged in a ring, we even get a CNS sequence using only 9 gates in 5 slices. Some of the resulting networks are given in Fig. 5.8.

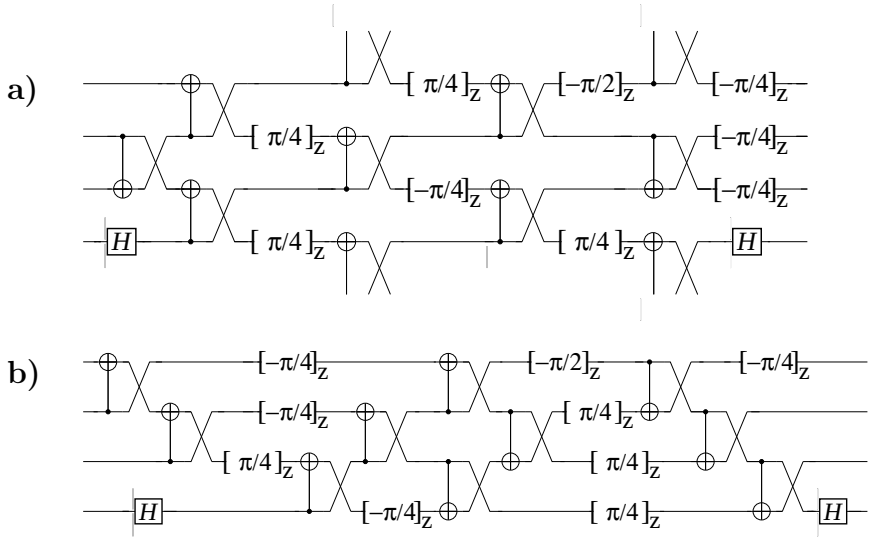


Figure 5.8: Programmable networks (actually, not any more), optimized for the CARRY gate. We give a parallelized CNS network for qubits arranged in a ring, and a non-parallel network for a linear setup of qubits. (The non-parallel network is still drawn with some operations in parallel. The point if talking of parallel or non-parallel networks is mainly concerned with the optimization—do we optimize the number of two-bit operations, or do we optimize the number of time slices?) Note the apparent—almost surprising—symmetry of the second network if rotated by 180 degrees.

5.6 Complexity considerations

We close the chapter with some considerations on complexity. The central issue in this discussion is the fact that on the one hand we complained about the growing difficulty if we want to extend existing classification schemes, on the other hand, our approach gives sequences of exponential length—$O(2^N)$, or $O(2^N/N)$ for parallel implementations. Therefore, one might feel tempted to ask whether programmable networks really give an advantage over the other approaches.

We start with some general considerations. Assume that only a limited number k of different basic operations is available. Then, for a sequence with length l, there exist k^l different possibilities (parallel execution excluded). On the other hand, the number of oracles goes as $2^{2^N}/N$ for large N. So if we want to be able to implement all oracles, most of the sequences will have a length of about

$$\begin{aligned} l &= \log\left[2^{2^N}/N\right] / \log k \\ &= \frac{1}{\log k}\left[2^N - \log N\right] \\ &\propto 2^N \end{aligned} \tag{5.25}$$

where $\log \equiv \log_2$.

We therefore see that the implementation of *all* Deutsch oracles necessarily

leads to sequences with a length exponential in N. Admittedly, the number of operations in our setup is not fixed, since the grid of the z rotations gets smaller as 2^{-N}. We will discuss this below.

Now let us compare the length of the programmable network to the length of the sequence for other approaches discussed at the beginning of this chapter. The implementation using the δs directly, i.e., via controlled phase flips needs 2^{N-1} ^{N}CPFs. Per ^{N}CPF, either $O(N)$ or $O(N^2)$ operations are needed [BBC$^+$95]. In the first case we need an auxiliary qubit which is not the case for our setup. In the second case, phase shifts with a grid $\propto 2^{-N}$ appear. Moreover, both complexities only hold for systems with arbitrary interactions. Therefore, in both cases an additional factor of N appears due to the SWAP operations (which moreover make the implementation more difficult to handle). Therefore, we end up with $O(N^3 2^N)$ operations if we work without ancillae.

For the implementation with controlled phase flips derived from the polynomial, the number of operations required in the worst case (i.e., for some polynomial containing many monomials) is $O(N^3 2^N)$ as well.[10] Hence, we see that both approaches lead to sequences with an additional factor of N^3 (or N^4 if we use parallelized networks) compared to the sequences obtained with programmable networks.

On the other hand, the simplification of these circuits leads back to a classification scheme. But according to (5.25) most of the sequences will still have exponential length after the simplification. Moreover, the number of classes will be double exponential unless we classify by an exponential number of invariants which makes the classification itself very hard. This means that we have to simplify a double exponential number of circuits each of which has exponential length. This work has to be done before the implementation itself. This is a serious obstacle if one considers the rapidly growing number of oracles or classes, cf. Table 5.1. Compared to this, if we use programmable networks there is only one fixed sequence which we have to simplify.

Finally, we would like to come back to the grid of the z rotations which clearly (cf. (5.17)) gets smaller as 2^{-N}. This could be a serious drawback since for growing qubit numbers we have to be able to control the one-bit operations more and more precisely. On the other hand, for large circuits a good control of the system is necessary in any case. Finally, this also leads to an observation how our programmable networks could be simplified. Recall that the relation between the rotation angles $\boldsymbol{\phi}$ of the z rotations and the phase shifts $\boldsymbol{\theta}$ applied by the network was

$$\boldsymbol{\theta} = \tfrac{1}{2} H_N \boldsymbol{\phi} \,. \tag{5.26}$$

[10] Note that $\displaystyle\sum_{k=2}^{N-1} \binom{N}{k} k^3 = -\left(N + N^3\right) + \left(\tfrac{3}{8}N^2 + \tfrac{1}{8}N^3\right) 2^N$.

From this we derived

$$\boldsymbol{\phi} = \tfrac{1}{2^{N-1}} H_N \boldsymbol{\theta} \; . \tag{5.27}$$

But in fact, this is not the only choice for $\boldsymbol{\phi}$ which satisfies (5.26). Since $\boldsymbol{\theta}$ is a vector of *phase angles* which are all 2π-periodic, we can indeed choose any $\boldsymbol{\phi}$ which yields the same *angle* $\boldsymbol{\theta}$. In other words, we can add any multiple of 2π to any component of $\boldsymbol{\theta}$, and then apply (5.27). Thereby, we get a large number of *different* $\boldsymbol{\phi}$s, due to the prefactor $1/2^{N-1}$ in (5.27).

This could be a way to avoid that the grid becomes exponentially small. One could even try to use this to set a subset of the ϕs to zero, so that the programmable network can be simplified similarly to the CARRY example, but without loosing its universal applicability to all Deutsch oracles.

However, it appears as if the ϕs we can set to zero were not the same for all balanced functions. Actually, if one tries to carry out this simplification one finds similarities to the existing classifications. This is an interesting subject for further investigation. We mention some ideas related to this in the appendix.

Chapter 6

Conclusions and outlook

The aim of this thesis was to show how the Deutsch–Jozsa algorithm can be implemented for more than three qubits on Josephson charge qubits, extending the work by Siewert and Fazio [SF01].

To this end, we started by analyzing the existing three-bit solutions. We could find the common principles underlying all of them, and gave a generalized description of the classification in terms of the polynomial representation of the oracle. We discussed different possible schemes how to extend the existing classifications, and demostrated that all these extensions cannot solve the problem of the rapidly growing number of classes.

These observations motivated the quest for an implementation which is not based on any classification. We could find a type of programmable networks which allow for the implementation of all Deutsch oracles on the same footing. These networks have a fixed operation sequence, where the only adjustable operations are one-bit z rotations. The relation between the Deutsch oracles and the rotation angles is straightforward.

We could devise different ways how to construct these networks. Particularly, we found a systematic recursive method how to build the networks for systems with only nearest neighbor coupling. For these systems we could also provide a parallelized construction which yields highly compact sequences. As a byproduct, we found a genuine two-bit operation for the XY interaction. In several cases, this CNS gate gives a considerable advantage over the conventional constructions using CNOT.

The programmable networks allow for the implementation of all Deutsch oracles on the same footing. Thus, we obtain an operation sequence of constant length, which is considerably shorter than the sequences derived by other methods which are similarly straightforward to apply. Although shorter sequences for at least a subset of the oracles could be found, this would require the optimization of each oracle separately, and therefore is not desirable.

The programmable networks derived are in fact multi-purpose networks. Beyond the oracles in Deutsch's algorithm, they can be used for the implementation

of Grover's oracle. Furthermore, together with the Hadamard transform these networks allow for the implementation of any controlled-NOT operation. This includes the Toffoli or multi-qubit CNOT gates as well as other gates with practical relevance such as the CARRY gate.

Still, there remain some interesting questions open which might be subject to future research. It is certainly interesting to ask whether there are other applications for the programmable networks, e.g., similar to the CARRY gate. It would be challenging to find out whether programmable networks which are similarly easy to apply can be found for non-diagonal matrices as well. To this end, one could try to use several programmable networks with Hadamard transforms placed inbetween, motivated by the observation that the programmable networks together with Hadamard are universal.

Another interesting open field is the investigation of the programmable networks with respect to errors. It would be interesting to know whether errors occuring in the network are attenuated or rather amplified. One might ask this question especially with respect to imprecisions of the z rotations. Finally, one might feel tempted to implement an error correction into the network. It seems especially appealing to try to integrate the five-bit error correcting code discussed in Chapter 4 into the programmable network. For a hardware setup with XY interaction and nearest neighbor coupling in a ring constructions which are similarly efficient as the one for the error correcting code alone are imaginable.

Finally, it would be challenging to investigate the effects caused by the 2π-periodicity of $\boldsymbol{\theta}$, and to find out whether this gives a straightforward method to further simplify the networks for the Deutsch oracle. This question seems to be related to the entanglement of the states generated by the Deutsch algorithm. The study of these relations might even lead to some new insight into the nature of the entanglement of these states. Some basic ideas on this are given in the appendix.

Appendix A

Formal equivalence of oracle representations

This appendix discusses some interesting aspects of the programmable networks as well as the polynomial representation of the oracles with respect to entanglement and locality. Recall the relation

$$\boldsymbol{\theta} = \tfrac{1}{2} H_N \boldsymbol{\phi} \; .$$

The N-bit Hadamard operation $H_N = H^{\otimes N}$ is a local operation. On the other hand, $\boldsymbol{\theta}$ is related to the oracle and the state generated by it in the Deutsch algorithm. Therefore, the question arises whether the entanglement properties of that state (which are not changed by local operations) can still be found in $\boldsymbol{\phi}$.

On the other hand, recall the polynomial representation. Define $\mathrm{mon}_{\mathbf{y}}(\mathbf{x})$ to be the monomial containing all x_i with $y_i = 1$. Then, the polynomial representation

$$f(\mathbf{x}) = \sum_{\mathbf{y}} p_{\mathbf{y}} \mathrm{mon}_{\mathbf{y}}(\mathbf{x})$$

defines a vector $\mathbf{p}$ of the coefficients of the monomials in the polynomial. The relation between $\mathbf{p}$ and $\boldsymbol{f}$ is clearly linear. But is it local as well?

Take a vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N) \equiv \sum_i \alpha_i |i\rangle$. This vector is separable (i.e., local) with respect to two subsystems if a decomposition

$$\sum_i \alpha_i |i\rangle = \sum_{j,k} \beta_j \gamma_k |j\rangle |k\rangle$$

in two vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ exists, i.e., $\alpha_{ij} = \beta_i \gamma_j$ with a combined index ij. Mathematically, $\boldsymbol{\alpha} = \boldsymbol{\beta} \otimes \boldsymbol{\gamma}$. Clearly, this is the definition of separability used in quantum mechanics, but actually it applies to any vector in any product space. An operation is local exactly if it preserves the locality of any vector (resp. state in quantum mechanics) in such a way that each of the parts of the result is already determined by the corresponding part of the input.

Now suppose $\boldsymbol{f}$ is separable with respect to two subsystems $\mathbf{x}^1, \mathbf{x}^2$ of $\mathbf{x}$, i.e., $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2)$. Then

$$\begin{aligned} f(\mathbf{x}) &= f^1(\mathbf{x}^1) f^2(\mathbf{x}^2) \\ &= \sum_{\mathbf{y}^1} p^1_{\mathbf{y}^1} \mathrm{mon}_{\mathbf{y}^1}(\mathbf{x}^1) \sum_{\mathbf{y}^2} p^2_{\mathbf{y}^2} \mathrm{mon}_{\mathbf{y}^2}(\mathbf{x}^2) \\ &= \sum_{\mathbf{y}^1, \mathbf{y}^2} p^1_{\mathbf{y}^1} p^2_{\mathbf{y}^2} \mathrm{mon}_{\mathbf{y}^1}(\mathbf{x}^1) \mathrm{mon}_{\mathbf{y}^2}(\mathbf{x}^2) \\ &= \sum_{\mathbf{y}^1, \mathbf{y}^2} p^1_{\mathbf{y}^1} p^2_{\mathbf{y}^2} \mathrm{mon}_{(\mathbf{y}^1, \mathbf{y}^2)}(\mathbf{x}^1, \mathbf{x}^2) \\ &= \sum_{\mathbf{y}^1, \mathbf{y}^2} p^1_{\mathbf{y}^1} p^2_{\mathbf{y}^2} \mathrm{mon}_{\mathbf{y}}(\mathbf{x}) \ . \end{aligned}$$

Therefore, the coefficients in the polynomial for $\boldsymbol{f}$ are $p_{\mathbf{y}} = p^1_{\mathbf{y}^1} p^2_{\mathbf{y}^2}$. Obviously, this means $\mathbf{p} = \mathbf{p}^1 \otimes \mathbf{p}^2$, where $\mathbf{p}^1$ and $\mathbf{p}^2$ are the polynomial representations of $\boldsymbol{f}^1$ and $\boldsymbol{f}^2$. Therefore, the mapping between $\boldsymbol{f}$ and $\mathbf{p}$ is local as well. The matrix of this mapping is $T_N = T^{\otimes N}$, where

$$T = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \ .$$

Together with the known relation $\boldsymbol{\theta} = \pi \boldsymbol{f}$, this gives the following diagram:

$$\begin{array}{ccc} \boldsymbol{f} & \xrightarrow{\cdot\pi} & \boldsymbol{\theta} \\ {\scriptstyle T_N}\downarrow & & \uparrow{\scriptstyle \frac{1}{2} H_N} \\ \mathbf{p} & & \boldsymbol{\phi} \end{array}$$

This means that the relation between all three different representations of f is local—a very interesting observation with respect to entanglement. Still, this is not the entanglement of the state created by the oracle. But by defining a new $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta} - \frac{\pi}{2}$, we see that $\hat{\theta}_{\mathbf{x}} = -\frac{\pi}{2}(-1)^{f(\mathbf{x})}$, and therefore $\hat{\boldsymbol{\theta}}$ has exactly the same entanglement and separability properties as the state generated by f. Correspondingly, we define $\hat{\boldsymbol{f}}$, $\hat{\boldsymbol{\phi}}$ and $\hat{\mathbf{p}}$ by the following diagram:

$$\begin{array}{ccccccc} \boldsymbol{f} & \xrightarrow{\cdot\pi} & \boldsymbol{\theta} & \xrightarrow{-\pi/2} & \hat{\boldsymbol{\theta}} & \xleftarrow{\cdot\pi} & \hat{\boldsymbol{f}} \\ {\scriptstyle T_N}\downarrow & & \uparrow{\scriptstyle \frac{1}{2} H_N} & & \uparrow{\scriptstyle \frac{1}{2} H_N} & & \downarrow{\scriptstyle T_N} \\ \mathbf{p} & & \boldsymbol{\phi} & & \hat{\boldsymbol{\phi}} & & \hat{\mathbf{p}} \end{array}$$

From this diagram, we can derive the following relation between $\hat{\boldsymbol{\phi}}$ and $\boldsymbol{\phi}$:

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \left(\tfrac{1}{2}H_N\right)^{-1}\left(I-\tfrac{\pi}{2}\right)\tfrac{1}{2}H_N\boldsymbol{\phi}\\ &= \boldsymbol{\phi}-\left(\tfrac{1}{2}H_N\right)^{-1}\tfrac{\pi}{2}\\ &= \boldsymbol{\phi}-\pi\delta_{\mathbf{0}}\ ,\end{aligned}$$

where $\delta_{\mathbf{0}}$ is the vector $(1,0,\ldots,0)$. (This can be checked easily, for instance by noting that $(1,\ldots,1)=(1,1)^{\otimes N}$, and $H(1,1)=(2,0)$). This means that $\hat{\boldsymbol{\phi}}$ and $\boldsymbol{\phi}$ only differ by the $\mathbf{0}$ component—which is the only component which does *not* appear in the programmable network. (On might notice that this is of course exactly the global phase we added before.) Similarly, one finds

$$\hat{\mathbf{p}}=\mathbf{p}-\tfrac{1}{2}\delta_{\mathbf{0}}\ ,$$

i.e., $\hat{\mathbf{p}}$ and $\mathbf{p}$ only differ in the prefactor of the constant monomial, which once more only contributes a global phase and does not enter the implementation derived from the polynomial. (In fact, things are a bit more complicated in this case, but it still works.)

To conclude, the entanglement properties of the state after the application of f are found in $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{f}}$, and therefore due to the locality of H_N and T_N in $\hat{\boldsymbol{\phi}}$ and $\hat{\mathbf{p}}$ as well. On the other hand, $\hat{\boldsymbol{\phi}}$ and $\hat{\mathbf{p}}$ can be used for the implementation of the oracle the same way $\boldsymbol{\phi}$ or $\mathbf{p}$ can be used. Therefore, the properties of the state concerning locality or entanglement can be found in the parameters of the programmable network or in the sequence of controlled phase flips derived from the polynomial.

This suggests that we can learn more about the entanglement by studying the different representations of the oracles, but one might as well suspect that this is exactly the reason why the simplification using the 2π-periodicity probably will fail. Finally, it tells us what the nice thing about the programmable networks is. These networks in fact do not reduce the complexity of the problem, but they are its optimal representation in the sense that all flexible parts are one-bit operations, and therefore it is most easy to adapt to its specific purpose.

Appendix B

Deutsche Zusammenfassung (German abstract)

Lange Zeit wurde davon ausgegangen, daß die Informatik, ähnlich der Mathematik, auf einer rein theoretischen Ebene betrieben werden könne, ohne die konkrete physikalische Realisierung des Computers zu berücksichtigen. Den ersten Hinweis darauf, daß diese Sichtweise nicht zutreffend ist, gab Landauer [Lan61], der nachweisen konnte, daß in einer irreversiblen Berechnung jeder Rechenschritt ein Mindestmaß an Entropie erzeugt, das von grundlegenden thermodynamischen Gesetzen bestimmt wird.

In der Folgezeit wurde immer klarer, daß es nicht möglich ist, Computer unabhängig von ihrer physikalischen Realisierung zu beschreiben: zum einen werden den Möglichkeiten von Computern durch die zugrundeliegenden physikalischen Theorien Grenzen gesetzt, genauso aber können neue physikalische Theorien auch neue Möglichkeiten im Bereich der Computer eröffnen. In dieser Hinsicht ist es interessant festzustellen, daß die traditionellen Computermodelle der Informatik vollständig im Rahmen der klassischen Physik beschreibbar sind.

Feynman [Fey82] wies als erster darauf hin, daß die Quantenmechanik neue Möglichkeiten im Bereich der Computer eröffnen könnte: während die Simulation eines quantenmechanischen Systems auf einem klassischen Computer im Normalfall exponentieller Resourcen bedarf, kann das System sich selbst in linearer Zeit „simulieren“.

Das erste konkrete Beispiel einer Aufgabenstellung, bei der ein quantenmechanisches System (ein „Quantencomputer“) einem klassischen Computer nachweisbar überlegen ist, kam vom Deutsch [Deu85]. Von diesem Zeitpunkt an war klar, daß Quantencomputer tatsächlich eine neue Art von Computern darstellen. Einige Jahre später gelang es Deutsch und Jozsa [DJ92], den Algorithmus so zu erweitern, daß sich ein exponentieller Geschwindigkeitsvorteil gegenüber klassischen Computern ergibt.

Diese Entdeckung beförderte die Forschung im noch jungen Gebiet des Quantencomputing. Eine Reihe von neuen Algorithmen wurde gefunden, von denen

insbesondere der Algorithmus von Shor [Sho94] zum Faktorisieren großer Zahlen und der Algorithmus von Grover [Gro96] zum Durchsuchen unstrukurierten Datenbanken zu nennen sind.

Die Entdeckung dieser Algorithmen, die praktische Perspektiven für die Anwendung von Quantencomputern aufzeigten, gab auch der Suche nach physikalischen Implementierungen neuen Auftrieb. Die ersten Experimente wurden mittels Kernspinresonanz in organischen Flüssigkeiten durchgeführt, da diese Systeme gute Eigenschaften bezüglich Kohärenz und kontrollierbarer Dynamik besitzten. Andererseits sind diesen Systemen aufgrund der zugrundeliegenden Physik in Bezug auf ihre Skalierbarkeit, die für die praktische Anwendbarkeit von Quantencomputern unabdingbar ist, klare Grenzen gesetzt.

In dieser Hinsicht scheinen Realisierungen auf Festkörperbasis wesentlich geeigneter zu sein, da hier der Skalierung keine prinzipiellen Hindernisse im Weg stehen. Eine Vielzahl von Implementierungen wurde vorgeschlagen, basierend unter anderem auf Spins in Quantendots, auf Kernspins sowie auf mesoskopischen supraleitenden Systemen. In dieser Arbeit beschäftigen wir uns mit einer speziellen Art supraleitender Systeme, nämlich den Josephson–Ladungsqubits.

Für viele dieser Implementierungen diente der Deutsch–Jozsa-Algorithmus als eine erste Demonstration für die Anwendbarkeit dieser Systeme als Quantencomputer. Collins *et al.* [CKH98] konnten nachweisen, daß der Deutsch–Jozsa-Algorithmus tatsächlich einen aussagekräftiger Test für die Realisierbarkeit von Quantencomputern mit einer spezifischen Implementierung darstellt, da er wesentlichen Gebrauch von den beiden Kernelementen von Quantenalgorithmen, Parallelität und Verschränkung, macht. Sie führten eine auf diese Elemente reduzierte Version des Algorithmus ein und konnten nachweisen, daß der Deutsch–Jozsa-Algorithmus erst von einer Anzahl von mindestens drei Quantenbits an tatsächlich Gebrauch von Verschränkung macht; daraus folgerten sie, daß der Algorithmus erst von drei Qubits aufwärts als ein echter Test für eine Implementierung angesehen werden kann.

Trotz der Entdeckung bei weitem komplexerer Quantenalgorithmen dient der Deutsch–Jozsa-Algorithmus immer noch in vielen Fällen als erster Test für Quantencomputing-Systeme, wohl auch, weil er vergleichsweise einfach aufgebaut ist. Aussagekräftige Implementierungen, also Implementierungen für drei Bits, existieren für Kernspinresonanzsysteme und für mittels SQUIDs gekoppelte Josephson-Ladungsqubits. Andererseits existieren keinerlei Implementierungen für mehr als drei Qubits. Dies liegt jedoch nicht, wie man meinen könnte, daran, daß Quantencomputer mit mehr als drei Qubits nach derzeitigem Stand nicht machbar wären, denn tatsächlich existieren Implementierungen mit bis zu sieben Qubits [VSB$^+$01]. Somit stellt sich die Frage nach der Möglichkeit einer Implementierung des Deutsch–Jozsa-Algorithmus für vier und mehr Qubits.

Ziel dieser Arbeit ist es aufzuzeigen, wie sich der Deutsch–Jozsa-Algorithmus mit mehr als drei Qubits realisieren läßt. Insbesondere soll damit eine Arbeit von Siewert und Fazio [SF01] fortgesetzt werden, die eine Drei-Bit-Implementierung

für mittels SQUID-loops gekoppelte Josephson-Ladungsqubits vorgeschlagen haben. Obwohl der Schwerpunkt dieser Arbeit auf dieser Implementierung liegt, sind die meisten Ergebnisse nicht darauf beschränkt.

Die Kapitel 2 und 3 geben eine Einführung in die Grundlagen des Quantencomputing im Allgemeinen sowie in die Physik der Josephson-Ladungsqubits. Kapitel 4 verknüpft die Ergebnisse dieser beiden Kapitel. Desweiteren führen wir ein neues Quantengatter ein, das für die verwendete Implementierung besonders geeignet ist und in vielen Fällen wesentlich kürzere Implementierungen als die klassischen Verfahren ermöglicht. Diese Ergebnisse wurden bereits anderweitig veröffentlicht [SS02b].

Kapitel 5 schließlich enthält die wesentlichen Ergebnisse der Arbeit. Vorläufige Resultate dieses Kapitels wurden bereits veröffentlicht [SS02a]. Wir beginnen mit einer Diskussion der exisiterenden Drei-Bit-Implementierungen. Dabei leiten wir eine Verallgemeinerung der verschiedenen Ansätze her, die auf einer Darstellung der verschiedenen beim Deutsch–Jozsa-Algorithmus zu implementierenden Funktionen durch Polynome beruht. Alle diese Ansätze gehen so vor, daß sie die verschiedenen Funktionen nach gewissen Merkmalen klassifizieren und dann untersuchen, wie man die Funktionen der verschiedenen Klassen implementieren kann. Wir diskutieren verschiedene Vorschläge, wie man, ausgehend von der Darstellung durch Polynome, die existierenden Klassifizierungen erweitern kann, und weisen nach, daß für all diese Ansätze die Anzahl der verschiedenen Klassen äußerst schnell wächst, so daß ein Klassifikationsansatz letztendlich nicht weiterführt.

Es scheint vorteilhafter zu sein, einen vereinheitlichten Ansatz zur Implementierung dieser Funktionen zu verwenden, bei dem alle Funktionen auf die selbe Weise realisiert werden können, so daß die langwierige Klassifikation der Funktionen entfällt. Wir leiten dafür das Konzept programmierbarer Netzwerke her. Diese Netzwerke ermöglichen es uns, alle Funktionen mit einem festen Netzwerk zu implementieren. Die gewünschte Funktion kann durch die Einstellung von einigen Einbit-z-Rotationen ausgewählt werden, wobei die Rotationswinkel durch eine einfache Operation aus der Funktion hergeleitet werden können.

Anschließend zeigen wir verschiedene Möglichkeiten auf, diese Netzwerke zu implementieren, unter anderem auch eine systematische rekursive Konstruktion sowie eine besonders effiziente parallele Version für die verwendete Implementierung. Bei diesen Netzwerken kann die im Kapitel 4 hergeleitete neue Zwei-Bit-Operation gewinnbringend eingesetzt werden.

Die Fähigkeit der programmierbaren Netzwerke, alle Funktionen des Deutsch–Jozsa-Algorithmus gleichermaßen zu implementieren, hat viele Vorteile: alle Sequenzen haben die gleiche Länge, und da wir nur eine Sequenz für alle verschiedenen Fälle benötigen, lohnt sich die Optimierung dieser Netzwerke besonders. Desweiteren können beispielsweise auch Fehlerkorrekturverfahren speziell auf das Netzwerk zugeschnitten werden.

Diese Netzwerke sind, abgesehen vom Deutsch–Jozsa-Algorithmus, auch noch

für eine Reihe weiterer Anwendungen geeignet: so kann man mit ihnen auch alle möglichen Datenbankfunktionen des Algorithmus von Grover implementieren, außerdem kann man mittels eines Tricks beliebige kontrollierte NOT-Operationen ausführen. In diese Klasse fallen auch eine Reihe von Operationen, die in praxisrelevanten Netzwerken benötigt werden, wie z.B. das Toffoli-Gatter oder das CARRY-Gatter, die unter anderem im Algorithmus von Shor Verwendung finden.

Zusammenfassend läßt sich feststellen, daß wir ein Verfahren aufzeigen konnten, mit dem sich eine Vielzahl von auf den ersten Blick sehr verschiedenen Operationen mit einem einzigen, einfach anzupassenden programmierbaren Netzwerk implementieren läßt. Diese programmierbaren Netwerke lassen sich auch auf Hardwaresystemen mit spezifischen Beschränkungen effizient implementieren und bieten somit eine breite Palette von Anwendungen.

Bibliography

[ADK01] Arvind, K. Dorai and A. Kumar, *Quantum entanglement in the NMR implementation of the Deutsch-Jozsa algorithm*, Pramana, Journal of Physics **56**, L705 (2001), quant-ph/9909067.

[AL99] D. S. Abrams and S. Lloyd, *Quantum Algorithm Providing Exponential Speed Increase for Finding Eigenvalues and Eigenvectors*, Phys. Rev. Lett. **83**, 5162–5165 (1999), quant-ph/9807070.

[Bar95] A. Barenco, *A universal two-bit gate for quantum computation*, Proc. R. Soc. Lond. A **449**, 679–683 (1995), quant-ph/9505016.

[BB01] J.-L. Brylinski and R. Brylinski, *Universal quantum gates*, (2001), quant-ph/0108062.

[BBC+95] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin and H. Weinfurter, *Elementary gates for quantum computation*, Phys. Rev. A **52**, 3457–3467 (1995), quant-ph/9503016.

[BBHT98] M. Boyer, G. Brassard, P. Høyer and A. Tapp, *Tight bounds on quantum searching*, Fortschr. Phys. **46**, 493–505 (1998), quant-ph/9605034.

[BDD+02] M. J. Bremner, C. M. Dawson, J. L. Dodd, A. Gilchrist, A. W. Harrow, D. Mortimer, M. A. Nielsen and T. J. Osborne, *Practical Scheme for Quantum Computation with Any Two-Qubit Entangling Gate*, Phys. Rev. Lett. **89**, 247902 (2002), quant-ph/0207072.

[Bea02] S. Beauregard, *Circuit for Shor's algorithm using 2n+3 qubits*, (2002), quant-ph/0205095.

[Ben73] C. H. Bennett, *Logical reversibility of computation*, IBM J. Res. Develop. , 525–532 (1973).

[BGLA02] Z. Bihary, D. R. Glenn, D. A. Lidar and V. A. Apkarian, *An Implementation of the Deutsch-Jozsa Algorithm on Molecular Vibronic*

Coherences Through Four-Wave Mixing: a Theoretical Study, Chem. Phys. Lett. **360**, 459 (2002), quant-ph/0110041.

[BLDS99] G. Burkard, D. Loss, D. P. DiVincenzo and J. A. Smolin, *Physical optimization of quantum error correction circuits*, Phys. Rev. B **60**, 11404 (1999), cond-mat/9905230.

[BS97] S. L. Braunstein and J. A. Smolin, *Perfect quantum-error-correction coding in 24 laser pulses*, Phys. Rev. A **55**, 945 (1997), quant-ph/9604036.

[BV93] E. Bernstein and U. Vazirani, Quantum complexity theory, in *Proc. Twenty-Fifth Ann. ACM Symp. on the Theory of Computing*, ACM Press, 1993.

[CEMM98] R. Cleve, A. Ekert, C. Macchiavello and M. Mosca, *Quantum algorithms revisited*, Proc. R. Soc. Lond. **454**, 339–354 (1998), quant-ph/9708016.

[CGK98] I. L. Chuang, N. Gershenfeld and M. Kubinec, *Experimental Implementation of Fast Quantum Searching*, Phys. Rev. Lett. **80**, 3408 (1998).

[CKH98] D. Collins, K. Kim and W. Holton, *Deutsch-Jozsa algorithm as a test of quantum computation*, Phys. Rev. A **58**, R1663–1666 (1998), quant-ph/9807012.

[CKH$^+$00] D. Collins, K. W. Kim, W. C. Holton, H. Sierzputowska-Gracz and E. O. Stejskal, *NMR quantum computation with indirectly coupled gates*, Phys. Rev. A **62**, 022304 (2000), quant-ph/9910006.

[CL83] A. Caldeira and A. Leggett, *Quantum tunnelling in a dissipative system*, Ann. Phys. **149**, 374–456 (1983).

[CMT01] G. Ciaramicoli, I. Marzoli and P. Tombesi, *Realization of a quantum algorithm using a trapped electron*, Phys. Rev. A **63**, 052307 (2001).

[CY95] I. L. Chuang and Y. Yamamoto, *Simple quantum computer*, Phys. Rev. A **52**, 3489 (1995), quant-ph/9505011.

[Deu85] D. Deutsch, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proc. R. Soc. Lond. **400**, 97–117 (1985).

[Deu89] D. Deutsch, *Quantum computational networks*, Proc. R. Soc. Lond. A **425**, 73–90 (1989).

[DiV98] D. P. DiVincenzo, *Quantum gates and circuits*, Proc. R. Soc. Lond. A **454**, 261–276 (1998), quant-ph/9705009.

[DiV00] D. P. DiVincenzo, *The Physical Implementation of Quantum Computation*, Fortschr. Phys. **48**, 771–783 (2000), quant-ph/0002077.

[DJ92] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. R. Soc. Lond. **439**, 553–558 (1992).

[Dra00] T. G. Draper, *Addition on a Quantum Computer*, (2000), quant-ph/0008033.

[DS96] D. DiVincenzo and P. Shor, *Fault-Tolerant Error Correction with Efficient Quantum Codes*, Phys. Rev. Lett. **77**, 3260 (1996), quant-ph/9605031.

[EF02] V. L. Ermakov and B. M. Fung, *Experimental realization of a continuous version of the Grover algorithm*, accepted for publication in Phys. Rev. A (2002), quant-ph/0208145.

[EJ96] A. Ekert and R. Jozsa, *Quantum computation and Shor's factoring algorithm*, Rev. Mod. Phys. **68**, 733–753 (1996).

[EKW01] B.-G. Englert, C. Kurtsiefer and H. Weinfurter, *Universal unitary gate for single-photon two-qubit states*, Phys. Rev. A **63**, 032303 (2001), quant-ph/0101064.

[Euc] *Εὐκλείδες, Στοιχεῖα* (Euclid, *Elements (Book VII)*, about 300 B.C.).

[EWD+01] P. Echternach, C. P. Williams, S. C. Dultz, P. Delsing, S. Braunstein and J. P. Dowling, *Universal Quantum Gates for Single Cooper Pair Box Based Quantum Computing*, Quantum Computation and Information **1**, 143 (2001), quant-ph/0112025.

[Fen01] M. Feng, *Grover search with pairs of trapped ions*, Phys. Rev. A **63**, 052308 (2001), quant-ph/0102122.

[Fey82] R. P. Feynman, *Simulating physics with computers*, Int. J. Theor. Phys. **21**, 467–488 (1982).

[GD92] H. Grabert and M. Devoret, editors, *Single Charge Tunneling*, Plenum, New York, 1992.

[Gro96] L. K. Grover, A fast quantum mechanical algorithm for database search, in *Proceedings of the 28th Annual ACM Symposium of Theory of Computing*, 1996.

[Høy99] P. Høyer, *Conjugated operators in quantum algorithms*, Phys. Rev. A **59**, 3280–3289 (1999).

[IAB+99] A. Imamoḡlu, D. D. Awschalom, G. Burkard, D. P. DiVincenzo, D. Loss, M. Sherwin and A. Small, *Quantum Information Processing Using Quantum Dot Spins and Cavity QED*, Phys. Rev. Lett. **83**, 4204 (1999), quant-ph/9904096.

[JL02] R. Jozsa and N. Linden, *On the role of entanglement in quantum computational speed-up*, (2002), quant-ph/0201143.

[JM98] J. A. Jones and M. Mosca, *Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer*, J. Chem. Phys. **109**, 1648 (1998), quant-ph/9801027.

[JMH98] J. A. Jones, M. Mosca and R. H. Hansen, *Implementation of a quantum search algorithm on a quantum computer*, Nature **393**, 344 (1998).

[Joz98] R. Jozsa, *Quantum algorithms and the Fourier transform*, Proc. R. Soc. Lond. **454**, 323–337 (1998), quant-ph/9707033.

[Kan98] B. Kane, *A silicon-based nuclear spin quantum computer*, Nature **393**, 133 (1998).

[KBDW01] J. Kempe, D. Bacon, D. P. DiVincenzo and K. Whaley, *Encoded Universality from a Single Physical Interaction*, Quantum Computation and Information **1**, 33 (2001), quant-ph/0112013.

[Kit95] A. Kitaev, *Quantum measurements and the Abelian stabilizer problem*, (1995), quant-ph/9511026.

[KLL00] J. Kim, J.-S. Lee and S. Lee, *Implementing unitary operators in quantum computation*, Phys. Rev. A **61**, 032312 (2000), quant-ph/9908052.

[KLLC00] J. Kim, J.-S. Lee, S. Lee and C. Cheong, *Implementation of the refined Deutsch-Jozsa algorithm on a 3-bit NMR quantum computer*, Phys. Rev. A **62**, 022312 (2000), quant-ph/9910015.

[KMSW00] P. G. Kwiat, J. R. Mitchell, P. D. D. Schwindt and A. G. White, *Grover's search algorithm: An optical approach*, J. Mod. Opt. **47**, 257 (2000).

[Knu97] D. E. Knuth, *The Art of Computer Programming, Vol. 2 – Seminumerical Algorithms*, Addison-Wesley, ^{3}1997.

[Knu02] D. E. Knuth, *The Art of Computer Programming, Vol. 4, Pre-fascicle 2a*, (2002), Pre-fascicle availabe at http://www-cs-faculty.stanford.edu/~ knuth/fasc2a.ps.gz.

[KW02] M. Keyl and R. F. Werner, How to correct small quantum errors, in *Coherent evolution in noisy environments*, Springer Verlag, 2002, quant-ph/0206086.

[KY02] A. R. Kessel and N. M. Yakovleva, *Schemes of implementation in NMR of quantum processors and Deutsch-Jozsa algorithm by using virtual spin representation*, (2002), quant-ph/0206106.

[Lan61] R. Landauer, *Irreversibility and Heat Generation in the Computing Process*, IBM J. Res. Develop. **3**, 183 (1961), The article has been reprinted in [Lan00].

[Lan00] R. Landauer, *Irreversibility and Heat Generation in the Computing Process*, IBM J. Res. Develop. **44**, 261 (2000).

[LBF98] N. Linden, H. Barjat and R. Freeman, *An implementation of the Deutsch-Jozsa algorithm on a three-qubit NMR quantum computer*, Chem. Phys. Lett. **296**, 61 (1998), quant-ph/9808039.

[LCYY00] D. W. Leung, I. L. Chuang, F. Yamaguchi and Y. Yamamoto, *Efficient implementation of coupled logic gates for quantum computation*, Phys. Rev. A **61**, 042310 (2000), quant-ph/9904100.

[LD98] D. Loss and D. P. DiVincenzo, *Quantum computation with quantum dots*, Phys. Rev. A **57**, 120 (1998), cond-mat/9701055.

[LYL+01] G. L. Long, H. Y. Yana, Y. S. Lia, C. C. Tua, J. X. Taoa, H. M. Chena, M. L. Liue, X. Zhange, J. Luoe, L. Xiaoa and X. Z. Zenge, *Experimental NMR realization of a generalized quantum search algorithm*, Phys. Lett. A **286**, 121 (2001).

[Mak00] Y. Makhlin, *Nonlocal properties of two-qubit gates and mixed states and optimization of quantum computations*, (2000), quant-ph/0002045.

[MDAK01] T. S. Mahesh, K. Dorai, Arvind and A. Kumar, *Implementing logic gates and the Deutsch-Jozsa quantum algorithm by two-dimensional NMR using spin- and transition-selective pulses*, Journal of Magnetic Resonance **148**, 95 (2001), quant-ph/0006123.

[Mer02] S. Mertens, *Computational complexity for physicists*, Computing in Science & Engineering **4**, 31–47 (2002), quant-ph/0012185.

[MFM+00] R. Marx, A. Fahmy, J. Myers, W. Bermel and S. Glaser, *Approaching five-bit NMR quantum computing*, Phys. Rev. A **62**, 012310 (2000).

[MOL+99] J. E. Mooij, T. P. Orlando, L. Levitov, L. Tian, C. H. van der Wal and S. Lloyd, *Josephson Persistent-Current Qubit*, Science **285**, 1036–1039 (1999).

[MPG01] D. Mozyrsky, V. Privman and M. Glasser, *Indirect Interaction of Solid-State Qubits via Two-Dimensional Electron Gas*, Phys. Rev. Lett. **86**, 5112 (2001), cond-mat/0012470.

[MSS99] Y. Makhlin, G. Schön and A. Shnirman, *Josephson-junction qubits with controlled couplings*, Nature **398**, 305 (1999), cond-mat/9808067.

[MSS01] Y. Makhlin, G. Schön and A. Shnirman, *Quantum-state engineering with Josephson-junction devices*, Rev. Mod. Phys. **73**, 357–400 (2001), cond-mat/0011269.

[MXKL00] F. Mang, Z. Xiwen, G. Kelin and S. Lei, *Quantum computing by pairing trapped ultracold ions*, (2000), quant-ph/0011001.

[NC00] M. A. Nielsen and I. A. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

[NPT99] Y. Nakamura, Y. Pashkin and J. Tsai, *Coherent control of macroscopic quantum states in a single-Cooper-pair box*, Nature **398**, 786–788 (1999), cond-mat/9904003.

[OMT+99] T. P. Orlando, J. E. Mooij, L. Tian, C. H. van der Wal, L. S. Levitov, S. Lloyd and J. J. Mazo, *Superconducting persistent-current qubit*, Phys. Rev. B **60**, 15398 (1999), cond-mat/9908283.

[Pre98a] J. Preskill, Fault-tolerant quantum computation, in *Introduction in quantum information and computation*, edited by H.-K. Lo, S. Popescu and T. P. Spiller, Singapore, 1998, World Scientific, quant-ph/9712048.

[Pre98b] J. Preskill, *Lecture Notes for Physics 229: Quantum Information and Computation*, 1998, available at http://www.theory.caltech.edu/people/preskill/ph229.

[PYA+02] Y. A. Pashkin, T. Yamamoto, O. Astafiev, Y. Nakamura, D. V. Averin and J. S. Tsai, *Quantum oscillations in two coupled charge qubits*, submitted to Nature (2002), cond-mat/0212314.

[PZ99] J. P. Paz and W. H. Zurek, *Environment-Induced Decoherence and the Transition From Quantum to Classical*, (1999), quant-ph/0010011.

[SF01] J. Siewert and R. Fazio, *Quantum algorithms for Josephson networks*, Phys. Rev. Lett. **87**, 257905 (2001), cond-mat/0105169, An extended version of this article has been published in [SF02].

[SF02] J. Siewert and R. Fazio, *Implementation of the Deutsch-Jozsa algorithm with Josephson charge qubits*, J. Mod. Opt. **49**, 1245 (2002), quant-ph/0112135.

[SFPS00] J. Siewert, R. Fazio, G. Palma and E. Sciacca, *Aspects of qubit dynamics in the presence of leakage*, J. Low Temp. Phys. **118**, 795–804 (2000).

[Sho94] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, in *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, edited by S. Goldwasser, page 124, Los Alamitos, CA, 1994, IEEE Computer Society, quant-ph/9508027. An extended version appeared in [Sho97].

[Sho97] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput. **26**, 1484–1509 (1997), quant-ph/9508027.

[Sim94] D. R. Simon, On the power of quantum computation, in *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, edited by S. Goldwasser, page 116, Los Alamitos, CA, 1994, IEEE Computer Society, An extended version appeared in [Sim97].

[Sim97] D. R. Simon, *On the power of quantum computation*, SIAM J. Comput. **26**, 1474–1483 (1997).

[SS02a] N. Schuch and J. Siewert, *Implementation of the Four-Bit Deutsch-Jozsa Algorithm with Josephson Charge Qubits*, phys. stat. sol. (b) **233**, 482–489 (2002).

[SS02b] N. Schuch and J. Siewert, *A natural two-qubit gate for quantum computation using the XY interaction*, accepted for publication in Phys. Rev. A (2002), quant-ph/0209035.

[SSH97] A. Shnirman, G. Schön and Z. Hermon, *Quantum Manipulations of Small Josephson Junctions*, Phys. Rev. Lett. **79**, 2371–2374 (1997).

[Tak00] S. Takeuchi, *Experimental demonstration of a three-qubit quantum computation algorithm using a single photon and linear optics*, Phys. Rev. A **62**, 032301 (2000).

[TdVR02] C. M. Tesch and R. de Vivie-Riedle, *Quantum Computation with Vibrationally Excited Molecules*, Phys. Rev. Lett. **89**, 157901 (2002), quant-ph/0208025.

[VAC+02] D. Vion, A. Aassime, A. Cottet, P. Joyez, H. Pothier, C. Urbina, D. Esteve and M. Devorett, *Manipulating the quantum state of an electrical circuit*, Science **296**, 886–889 (2002), cond-mat/0205343.

[VAZ+01] J. Vala, Z. Amitay, B. Zhang, S. R. Leone and R. Kosloff, *Experimental Implementation of the Deutsch-Jozsa Algorithm for Three-Qubit Functions using Pure Coherent Molecular Superpositions*, (2001), quant-ph/0107058.

[VBE96] V. Vedral, A. Barenco and A. Ekert, *Quantum networks for elementary arithmetic operations*, Phys. Rev. A **54**, 147 (1996), quant-ph/9511018.

[VHC02] G. Vidal, K. Hammerer and J. I. Cirac, *Interaction cost of non-local gates*, Phys. Rev. Lett. **88**, 237902 (2002), quant-ph/0112168.

[VLS+01] A. S. Verhulst, O. Liivak, M. H. Sherwood, H.-M. Vieth and I. L. Chuang, *Non-thermal nuclear magnetic resonance quantum computing using hyperpolarized Xenon*, Appl. Phys. Lett. **79**, 2480 (2001), quant-ph/0105147.

[VSB+01] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood and I. L. Chuang, *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*, Nature **414**, 883–887 (2001), quant-ph/0112176.

[VSS+00] L. M. Vandersypen, M. Steffen, M. H. Sherwood, C. S. Yannoni, G. Breyta and I. L. Chuang, *Implementation of a three-quantum-bit search algorithm*, Appl. Phys. Lett. **76**, 646 (2000), quant-ph/9910075.

[VYW+00] R. Vrijen, E. Yablonovitch, K. Wang, H. W. Jiang, A. Balandin, V. Roychowdhury, T. Mor and D. DiVincenzo, *Electron-spin-resonance transistors for quantum computing in silicon-germanium heterostructures*, Phys. Rev. A **62**, 012306 (2000), quant-ph/9905096.

[YMB+02] F. Yamaguchi, P. Milman, M. Brune, J. M. Raimond and S. Haroche, *Quantum search with two-atom collisions in cavity QED*, Phys. Rev. A **66**, 010302(R) (2002), quant-ph/0203146.

[YMY00] F. Yamaguchi, C. P. Master and Y. Yamamoto, *Concurrent Quantum Computation*, (2000), quant-ph/0005128.

[YSV+99] C. S. Yannoni, M. H. Sherwood, L. M. Vandersypen, D. C. Miller, M. G. Kubinec and I. L. Chuang, *Nuclear Magnetic Resonance Quantum Computing Using Liquid Crystal Solvents*, Appl. Phys. Lett. **75**, 3563 (1999), quant-ph/9907063.

[ZLSD02] J. Zhang, Z. Lu, L. Shan and Z. Deng, *Experimental implementation of generalized Grover's algorithm of multiple marked states and its application*, (2002), quant-ph/0208102.

Acknowledgements

In the first place, I have to thank Jens Siewert for being my supervisor and introducing me to this fascinating new field of physics. I enjoyed a lot all the stimulating discussions we had. He always took the time for these discussions; most of them lasted four hours rather than the thirty minutes I had asked for. I am grateful for the freedom he gave me in my work on this thesis, and for believing in the relevance of my results sometimes more than I did. He encourged me to present my results at various conferences, and taught me a lot about how to give talks. He spent a lot of time as well on my first papers, patiently trying to teach me a concise style of writing (considering the length of this thesis, he obviously failed ;-). Proofreading my drafts usually resulted in rewriting the whole article. Also, I owe him thanks for trying to improve my English (hopefully, it is tolerable after all). Further on, although being very busy, he thoroughly proofread this whole thesis. Finally, I want to thank him for his help with finding a PhD position. I enjoyed a lot working with him.

Then, I have to thank Klaus Richter for taking me as a Diploma student, for supporting my work, for giving me the opportunity to participate in conferences, and of couse for the nice atmosphere at his chair.

At this point I want to thank all the members of the Richter chair who made my time here a very enjoyable one. I will certainly miss our daily coffee break with its chats and the birthday cakes, and especially the collective solving of the *Zeit* and *Sueddeutsche* crosswords on Thursdays and Fridays.

I also owe some special thank to our secretary Angie Reisser, who was a big help with all the unavoidable formalities. A lot of the good atmosphere at the chair is due to her.

There are several people I have to thank for stimulating discussions, and for confirming my belief that physics is fun. Especially, I would like to thank Andreas Osterloh for clarifying discussions about the nature of entanglement, and Dani Shapira for giving me some new insight to quantum search algorithms. I also want to thank all the people I met at conferences and who made these conferences enjoyable ones in any respect. Finally, I would like to thank Roland Hoffmann who was opened to discussions about physics (and to other enjoyable things, especially having cocktails) all the five years we studied together.

Moreover, I would like to thank all my friends, especially the people from the *Regensburger Studententheater*, for giving me some diversion from physics throughout my study, but especially throughout the work on this thesis.

Finally, I want to thank my parents for their continuous support and encouragement. Without them, achieving this goal would not have been possible.

Erklärung

Hiermit erkläre ich, daß ich die Diplomarbeit selbständig angefertigt und keine Hilfsmittel außer den in der Arbeit angegebenen benutzt habe.

Regensburg, den 19. Dezember 2002

(Norbert Schuch)