ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

STL SYSTEMTECHNIKLABOR

An Introduction to Quantum Computing

Joel Alexandre Vogt

Technical Report – STL-TR-2021-03 – ISSN 2364-7167

stl.htwsaar.de

Technische Berichte des Systemtechniklabors (STL) der htw saar
Technical Reports of the System Technology Lab (STL) at htw saar
ISSN 2364-7167

Joel Alexandre Vogt: An Introduction to Quantum Computing
Technical report id: STL-TR-2021-03

First published: April 2021
Last revision: April 2021
Internal review: André Miede

For the most recent version of this report see: https://stl.htwsaar.de/

Title image source: geralt, https://pixabay.com/de/photos/platine-leiterbahnen-schaltkreise-2440249/

htw saar – Hochschule für Technik und Wirtschaft des Saarlandes (University of Applied Sciences)
Fakultät für Ingenieurwissenschaften (School of Engineering)
STL – Systemtechniklabor (System Technology Lab)
Prof. Dr.-Ing. André Miede (andre.miede@htwsaar.de)
Goebenstraße 40
66117 Saarbrücken, Germany
https://stl.htwsaar.de

An introduction to Quantum Computing

Joel Alexandre Vogt
htw saar – Hochschule für Technik und Wirtschaft des Saarlandes
Seminar "Angewandte Informatik"
Wintersemester 2020

***Abstract*—This study provides an introduction to quantum computing and programming. Exploring the fundamentals of quantum computers, this research considers classic computing, how the basic unit changed the world, and how computers can compute with millions of data points every second using simple logic gates. Chapter 2 outlines properties of quantum computing known as superposition and entanglement. In the third and final chapter, simple circuits are shown to illustrate how quantum gates behave and a simple board game will be programmed.**

I. Classical Computing

A. The basic unit – binary digit

Classical computing is omnipresent. In today's world, nearly everybody owns a smartphone or a classic computer [1]. A classic computer is basically not more than a merging of simple logic gates that work with 0s or 1s. The digital world uses the binary system to communicate video and images, which means it is possible to see a live video of our family on the other side of the globe. All of this is possible thanks to the binary system formulated by Gottfried Wilhelm Leibniz (1664–1714), a German philosopher and mathematician. To perform fast calculations, a modern computer, such as the M1-Chip from Apple released in late 2020, uses 16 billion transistors to compute information with integrated circuits [2]. These circuits are constructed with logic gates and are discussed in the next subchapter. The transistors are connected in a particular way to enable the programmer to decide the sequence of logical operations. Numbers from 0 to 2^n – 1 can be represented with n binary digits (bits), but even with this level of computing power, a classical computer cannot, for example, find prime factors of logic numbers in an acceptable amount of time, as classical computers must test each possibility in series, while a quantum computer can perform the same task simultaneously.

B. The boolean function - logic gates

In the 19th century, George Boole explained that some parts of logic could be expressed in terms of algebra. We will show how boolean logic behaves by using truth tables and show how our computers use logic gates to perform calculations. Like real gates, a logic gate can either let the bit pass, or not. Simply said, "1" means that the voltage is high, while "0" means it is low. A 1 let it pass and 0 blocks it. To better understand quantum gates, it is necessary to watch what the classical logic gates do. The logical AND-Gate only shows a 1 if both of the entry points receive a bit with the voltage on "high." In C. Bernhardts book "Quantum Computing for everyone" it is explained that some of the classical logic gates are not reversible, that means for example, that the logic gate "AND" is not a reversible gate, because if we receive an output of 1, we know that both inputs have been 1, but an output of 0 is not letting us know which of the three input pairs (00, 01 or 10) was used to get that defined output [3].

Table I
AND - Gate

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The logical OR-Gate shows a 1 if at least one of the entry points receives a "high."

Table II
OR - Gate

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

The logical NOT-Gate inverts the signal. A 1 becomes a 0, and a 0 becomes a 1. In addition, this gate is also reversible, since we know exactly what input created the output.

Table III
NOT - Gate

A	Y
0	1
1	0

II. Properties Of Quantum Computing

At the microscopic level of electrons, the world is based on quantum mechanics. To understand why quantum bits (qubits) can provide a strong computational power, we must understand the properties of quantum computing discussed in the chapter Superposition and Entanglement. While explaining the full details of these properties is beyond the scope of the present work, an overview of the concepts is provided in this chapter.

A. Qubits

A qubit is the quantum analogue of a bit. While a bit can either be a one or zero, a qubit can be both at the same

time (the so-called superposition that is explored in the next section). A qubit is a two-level quantum system in which there can be $|1\rangle$ and $|0\rangle$ simultaneously. The notation $|\bullet\rangle$ is called the "ket-" or "Dirac-notation" and is a notation for a quantum state [4]. There's also the Bloch sphere to represent the state of a qubit:

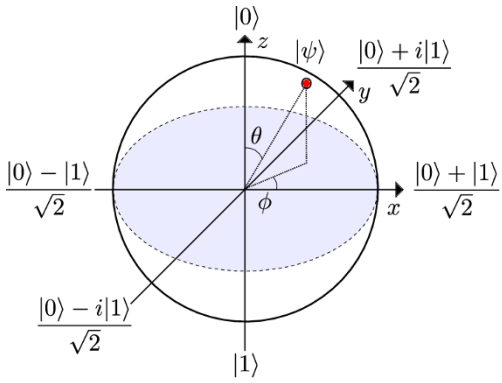

Figure 1. The Bloch sphere representation of a qubit [5]

With the help of the Bloch sphere, we can realize any state of a qubit with the help of vectors that begin at the middle of the sphere and end on the surface of it, where the poles represent 1 and 0. When a qubit is measured, it points to one of the two poles. For example, if the qubit points at the equator of the sphere, the chance that it will collapse to either 1 or 0 is 50%.

B. Superposition

As described in the Chapter before, superposition is the state of a qubit before it is measured. A qubit is in a state of $|1\rangle$ and $|0\rangle$ simultaneously until the measurement shows a $|1\rangle$ or a $|0\rangle$. The corresponding formula is $|\psi\rangle$=$\alpha|0\rangle$+$\beta|1\rangle$, which shows that, before being measured, a qubit has every state possible. To better understand this principle, we will use a property of light called polarization that illustrates a superposition of states. Since polarization has no preferred direction, we can only measure how light behaves through a polarizing filter, which is "a thin film with an axis that only allows light with polarization parallel to that axis to pass through" [6]. For example, we can represent a vertical polarization as $|\uparrow\rangle$ and a horizontal polarization as $|\rightarrow\rangle$. With these two states, we can form a basis for any polarization of light. Any polarization state, represented as $|\psi\rangle$, can be written as a linear combination of these states:

$$|\psi\rangle = \alpha\,|\uparrow\rangle + \beta\,|\rightarrow\rangle$$

Here, α and β are complex numbers called amplitudes. As shown in the formula above, α is linked to the vertical polarization and β with the horizontal polarization. Furthermore, if we select a vertical polarization, we can then use a second polarizing filter, which axis we orient at 45° (that would be $\nearrow$, because it is between $\uparrow$ and $\rightarrow$). Now the question would be, if we see any light that passes through the second filter? The answer is yes — and shows us how superposition works. Now, the superposition is:

$$|\uparrow\rangle = \frac{1}{\sqrt{2}}\,|\nearrow\rangle + \frac{1}{\sqrt{2}}\,|\nwarrow\rangle$$

The formula shows us that exactly $\frac{1}{2}$ of the light gets through the second filter. In Max Born's paper from 1926, he demonstrated "that the modulus squared of the amplitude of a state is the probability of that state resulting after measurement" [6], [7]. In the precedent case, since the amplitude is $\frac{1}{\sqrt{2}}$, the probability of this state is $\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$, which means that the probability of measuring the light in one of the two states is 50%. Furthermore, it is important that the sum of the squares of all the amplitudes in the superposition is equal to 1 [6], so that we have:

$$|\alpha|^2 + |\beta|^2 = 1$$

C. Quantum Entanglement

Quantum entanglement is the second property of quantum computing that we will look into. Usually, when we work with particles, we know that each own particle has a certain quantum state. But when two particles act as a system, we call them entangled. In 1935, Einstein, Podolsky and Rosen published a paper on quantum entanglement. That paper, also known as EPR [8], shows that two particles (i.e qubits) that are entangled with each other, will automatically trigger instantly a change of state of the other one — even if they are lightyears away from each other [9]. In other words, these states are not separable. The entanglement state is, after Albert Einstein, a "spooky action at distance" [10]. To get a grasp of it, we will use a simple example like in the previous subchapter. We assume that we have two entangled qubits and one is given to Alice and one to Bob. Alice and Bob are in two different rooms or even cities. The current state of both qubits is:

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Meanwhile, Alice and Bob measure their own qubits with a probability of $\frac{1}{2}$ that it will collapse into $|1\rangle$ or $|0\rangle$. Neither Alice or Bob knows if the other one already has measured their qubit, but both can see what result they got. But if they meet again, they will see that they have identical results. This is called a Bell-State, named after physicist John Bell (1928-1990). As followed, Bell created four states that explain how two qubits can be entangled [11]:

$$\phi^+ = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle),$$
$$\phi^- = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle),$$
$$\psi^+ = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle),$$
$$\psi^- = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$$

In this chapter, we have seen the two major quantum mechanics called superposition and entanglement. In the following chapter, we will dive into simple quantum programming.

III. Quantum programming with Qiskit

Qiskit is an open-source framework for working with quantum computers. It provides a set of tools for interacting with quantum systems and simulators [12]. This chapter explores some basic quantum circuits, explains their behavior and basic quantum programming.

A. Quantum Gates

In classical computers, logical operators, such as AND, OR, and NOT, are used to build complex computations. As quantum computers cannot use gates with the simple 1 or 0 logic, the so-called quantum gates must be used. The most significant difference between classical logic gates and quantum gates is reversibility. In classical gates such as OR, it is not known whether the output "1" comes from the input (01), (10), or (11). This information is lost at the moment the information passes the gate. Therefore, it is important to use reversible quantum gates, because the laws of quantum physics are reversible in time [13]. This section examines a quantum logic gate called "controlled NOT gate" (CNOT-Gate). The CNOT-Gate is "a conditional gate that performs an X-gate on the second qubit (target), if the state of the first qubit (control) is $|1\rangle$" [14]. In terms of classical bits, the CNOT-Gate is comprised of the following:

Input		Output	
x	y	a	b
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

Figure 2. The CNOT-Gate: (a) truth table, (b) reversible circuit representation. [15]

The upper input goes through the gate without any change and can be observed at the output. The important part is the control-module. If the top signal is $|0\rangle$, then the input $|y\rangle$ will not change; if the upper signal is $|1\rangle$, the gate will reverse the bottom signal. Since it's similar to an XOR gate, we use the symbol $\oplus$ [16].

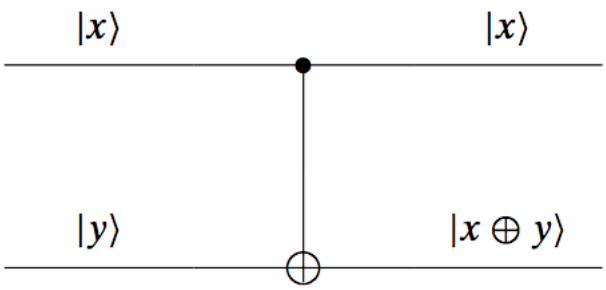

Figure 3. The controlled-NOT gate [16]

B. A half adder using Qiskit

A half adder is the addition of two bits using a combination circuit. The circuit will have a part that encodes the input, a part that executes the algorithm, and a part that extracts the result [17].

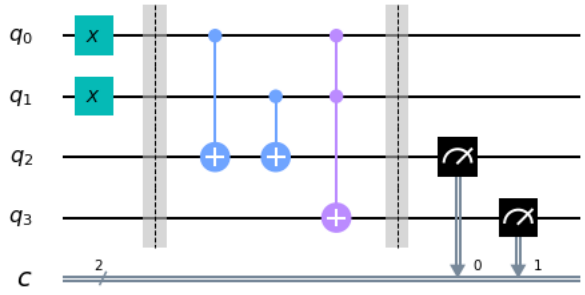

Figure 4. Half adder using Qiskit [17]

In this image, the two bits we want to encode as input are stored q0 and q1. This example encodes a "1" in both qubits and wants to compute 1+1. The result will show two bits that are output by q2 and q3.

C. Creating a game using Qiskit

The following code will mostly be outdated in the near future since quantum programming is still in the early stages of development. We will now create a battleship game using Qiskit, which is embedded in Python. The Python part will deal with the "normal" parts of the program and Qiskit will use the SDK for the quantum part. We will skip some parts that are irrelevant such as handler and automated processes. This section will mainly use the guide from Dr James Wootton, who is an IBM researcher [18].

Firstly, we have to create an account on IBM's Quantum Experience portal and import following modules to be able to run the code:

```
from qiskit import ClassicalRegister,
    QuantumRegister
from qiskit import QuantumCircuit, execute
```

Listing 1. Import modules for IBM's Quantum Experience

Secondly, we register the API to be able to run the program on IBM's quantum device.

```
from qiskit import register

import Qconfig
qx_config = {
    "APItoken": Qconfig.APItoken,
    "url": Qconfig.config['url']}

register(qx_config['APItoken'], qx_config['url'])
```

Listing 2. Register API

Now, we will implement a yes or no choice that let the player chose if a real five qubit quantum processor or just a simulator will be used. If the answer is "y" the real device is

chosen, if the answer is ”n”, the simulator will be taken into account.

```
device = ask_for_device()

from qiskit import Aer, IBMQ
try:
    backend = Aer.get_backend(device)
except:
    backend = IBMQ.get_backend(device)
```

Listing 3. Ask which device should be used

Since a quantum computer needs to calculate statistics on the possibly random outputs, we have to tell the device how many rounds it has to go through.

```
shots = 1024
```

Listing 4. Set the amount of rounds

Our next step is setting up a game board with five points, where each player can choose where to put three ships. Each point corresponds to the five qubits that the IBM device has. For example, position 3 is also the third qubit (q[3]) on the actual machine. The board is visualized as followed:

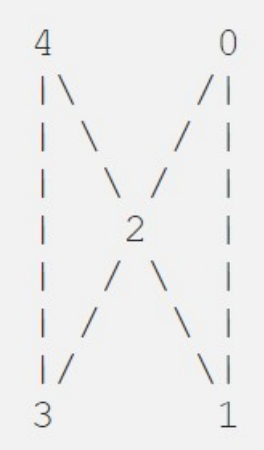

Figure 5. The battleship game board [18]

We use ”shipPos” as entry points for each player, where their choices are stored (player=0 and player=1) and for each of the ships they place on the game board. For example, if player 2 puts one of his ships on position 1, the entry will be: shipPos[1][0]. We also create a bomb[player][position] in the main game loop to count how many times a player has bombed that position in the game session.

Now, we will implement two quantum programs that need to run in each round. Each is simulating the grid of one player:

```
qc = []
for player in range(2):
    q = QuantumRegister(5)
    c = ClassicalRegister(5)
    qc.append( QuantumCircuit(q, c) )
```

Listing 5. Simulation of the grids

The array ”qc” is used to store information about the game. Also, each program is initialized with a register of five qubits and bits to be able to cover the whole grid. To address a grid of a player, we can use qc[player]. For every bomb that a player threw, we add a line to qc[player]. Also, this Japanese variant of the Battleship game has the following rule: One ship can be sunk by a single bomb, one that sinks after two bombs and one after three. To simulate this, we have to put a 0 on ships that are intact and a 1 that has been destroyed. We will apply a NOT on the ship that can be sunk with one bomb. To demonstrate this, we use the following script that creates a ship, sink it and then look if it is still intact:

```
q = QuantumRegister(1) # initialize a register with
    a single qubit
c = ClassicalRegister(1) # initialize a register
    with a normal bit
qc = QuantumCircuit(q, c) # create and empty quantum
     program
qc.u3(math.pi,0,0, q[0]) # apply a NOT to the qubit
qc.measure( q[0], c[0] ) # measure the qubit
```

Listing 6. Script for the NOT application

Back to our game with the two players. With the following line, we apply a NOT to a qubit where ”frac” and ”position” have to replaced by numbers:

```
qc[player].u3(frac * math.pi, 0.0, 0.0, q[position])
```

Listing 7. Apply a NOT to a qubit

To apply the Japanese rule, a ”half a NOT” will be applied to the second ship and a ”third of a NOT” to the last ship. To simulate this rule, we use frac = 1/(ship+1).

When the bombing is over, we can see what happened with the following code:

```
for qubit in range(5):
    qc[player]t.measure(q[qubit], c[qubit])
```

Listing 8. Show the result of the bombing

Now, we can run that pair of quantum programs, where ”backend” was the device chosen by the player at the start of the game:

```
job = execute(qc, backend, shots=shots)
```

Listing 9. Run the quantum programs

When the program is correctly submitted, we can extract the game data, where the results are copied over to grid:

```
for player in range(2):
    grid[player] = job.result().get_counts(qc[player
    ])
```

Listing 10. Extract the game data

Since qubits are noise-sensible, we cannot expect a 100% damage to a ship to count it as sunken. So in our game, a ship is sunken when the damage is greater than 95%. To interpret the results, we can assume the following: A player put their third ship at position 2 and the other player bombed it. The result will look like: {’00000’: 734, ’00100’: 290}. Meaning that 290 times of the 1024 runs (chosen at the beginning), the ship at position 2 suffered some damage.

Once the damage is calculated by the program, the game board can be represented as:

Figure 6. The battleship game board after the bombing [18]

In the figure above, we see that only the ship at position 2 has been hit and suffered 50% damage. The game now starts again the loop until one of the players has lost all of its ships.

IV. Summary

In this paper, we have seen how the digital world is based on billions of transistors and on two numbers: 1 and 0. Also, we now know how qubits can behave and play by their own rules thanks to superposition and entanglement. In the last Chapter, we created a simple board game with five real qubits from a device created by IBM. Quantum computing offers many exciting things and surely some risks to our current techniques of cryptography.

References

[1] R. Sutor, *Dancing with qubits : how quantum computing works and how it can change the world*. Birmingham, UK: Packt Publishing, Ltd, 2019.

[2] Apple Inc., "Small chip. giant leap." [Online]. Available: https://www.apple.com/mac/m1/

[3] C. Bernhardt, *Quantum computing for everyone*. Cambridge, Massachusetts: The MIT Press, 2019.

[4] P. Dirac, "A new notation for quantum mechanics," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 35, no. 3, pp. 416–418, jul 1939.

[5] A. Kockum, *Quantum optics with artificial atoms*. Gothenburg: Department of Microtechnology and Nanoscience, Chalmers University of Technology, 2015.

[6] J. Hidary, *Quantum computing : an applied approach*. Cham: Springer, 2019.

[7] M. Born, "Quantenmechanik der Stoßvorgänge," *Zeitschrift für Physik*, vol. 38, no. 11, pp. 803–827, 1926.

[8] A. Einstein, B. Podolsky, and N. Rosen, "Can quantum-mechanical description of physical reality be considered complete?" *Physical Review*, vol. 47, no. 10, pp. 777–780, may 1935.

[9] A. Kyrillidis, "Introduction to quantum computing: The notion of entanglement." [Online]. Available: http://akyrillidis.github.io/notes/quant_post_5

[10] A. Einstein, H. Born, and M. Born, *Briefwechsel 1916-1955*, ser. rororo Taschenbücher. Rowohlt, 1972, p. 254. [Online]. Available: https://books.google.de/books?id=LQIsAQAAIAAJ

[11] M. Homeister, *Quantum Computing verstehen*. Springer-Verlag GmbH, Oct. 2018. [Online]. Available: https://www.ebook.de/de/product/33440904/matthias_homeister_quantum_computing_verstehen.html

[12] Qiskit, "Open-source quantum development." [Online]. Available: https://qiskit.org/

[13] T. Hey, "Quantum computing: an introduction," *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 105–112, 1999.

[14] Qiskit, "The cnot-gate." [Online]. Available: https://qiskit.org/textbook/ch-gates/multiple-qubits-entangled-states.html#cnot

[15] A. Moustafa, A. Younes, and Y. F. Hassan, "A customizable quantum-dot cellular automata building block for the synthesis of classical and reversible circuits," *The Scientific World Journal*, vol. 2015, pp. 1–9, 2015.

[16] A. Kyrillidis, "The controlled-not gate." [Online]. Available: http://akyrillidis.github.io/notes/quant_post_6

[17] Qiskit, "Adding with qiskit." [Online]. Available: https://qiskit.org/textbook/ch-states/atoms-computation.html

[18] J. Wootton, "How to program a quantum computer." [Online]. Available: https://medium.com/qiskit/how-to-program-a-quantum-computer-982a9329ed02

All online sources and references were last accessed on March 08, 2021.

LISTINGS