The Little Far-Out Book

a summary by Dan-Adrian German for the
6W2 Summer 2019 QC Y390s (... and beyond)

"Through the hoop, Bob! Through the hoop!"

Last update: Tuesday 16th July, 2019 at 23:18
`https://www.cs.indiana.edu/~dgerman/little-fob/draft.pdf`

Contents

Introduction

What is a QPU?	It's a Quantum Processing Unit.
Can you show me how to access one?	Yes, this is a hands-on book.
Where is the QPU?	`https://oreilly-qc.github.io/`
Do I need to know Javascript?	Very little.
What else should I be familiar with?	Trigonometric functions.
Do I need to know Calculus?	A few basic things.
Will we use complex numbers?	It's good to know what they are.
Tell me something I don't expect.	A QPU is in fact a co-processor.
Is there a set of primitive operations?	We'll get to those in due time.
Where do we start?	A few fundamental concepts.
What is a qubit?	That's a good place to start.

Can we visualize a qubit?	Before or after a readout?
Why is there a difference?	Because after the readout of a qubit, you'll always get a 0 or a 1.
So in that regard a qubit is just a bit.	Yes. Before readout, however, qubits can exist in a superposition of states.
Can you show me?	Take a look.

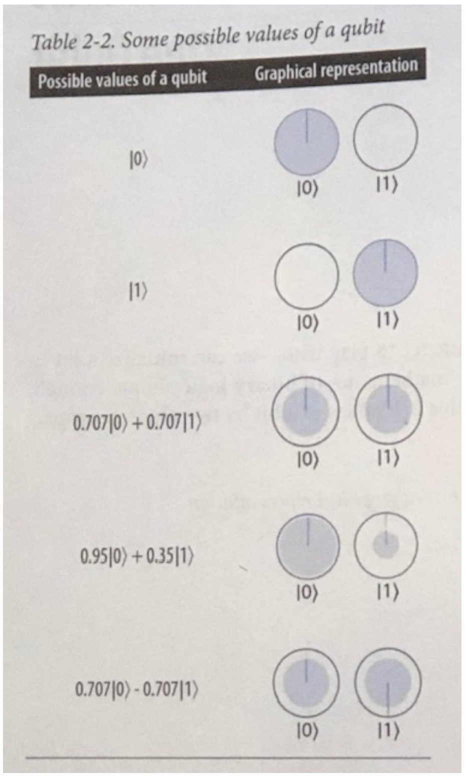


Table 2-2. Some possible values of a qubit

Possible values of a qubit	Graphical representation
$\|0\rangle$	$\|0\rangle$ $\|1\rangle$
$\|1\rangle$	$\|0\rangle$ $\|1\rangle$
$0.707\|0\rangle + 0.707\|1\rangle$	$\|0\rangle$ $\|1\rangle$
$0.95\|0\rangle + 0.35\|1\rangle$	$\|0\rangle$ $\|1\rangle$
$0.707\|0\rangle - 0.707\|1\rangle$	$\|0\rangle$ $\|1\rangle$

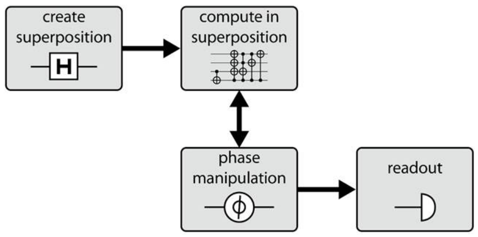
Can you explain the coefficients?	$a^2 + b^2 = 1$
And what is: $a\|0\rangle + b\|1\rangle$?	A qubit in superposition.

Organization

This book is a hands-on programmer's guide to using quantum computing technology. A tried-and-tested approach for getting hands-on with new programming paradigms is to learn a set of conceptual primitives. The heart of this book focuses on building an intuition for a set of quantum primitives—ideas forming a toolbox of building blocks for problem-solving with a QPU. To prepare you for these primitives, we first introduce the basic concepts of qubits (the rules of the game, if you will). Then, after outlining a set of Quantum Processing Unit (QPU) primitives, we show how they can be used as building blocks within useful QPU applications.

What exactly is a qubit? How can we visualize one? How is it useful? These are short questions with complicated answers. In the first part of the book we'll answer these questions in a practical way. We begin in Chapter 1 by first describing and making use of a single qubit. The additional complexity of multi-qubit systems are then covered in Chapter 2. Along the way we'll encounter a number of single and multi-qubit operations, and Chapter 3 puts these straight to use, describing how to perform quantum teleportation. Bear in mind that the code samples that punctuate our discussion can be run on the QCEngine simulator that was already mentioned[1] using the provided links.

Once we know how to describe and manipulate qubits at a basic level, we can introduce some higher-level QPU *primitives*. These primitives form a toolbox, ultimately allowing us to build full-blown applications. Very roughly speaking, quantum applications tend to have the structure shown below:



QPU primitives help us to fill out this structure[2]. Primitives associated with the second of these four steps (*compute in superposition*) allow us to compute using the implicit parallelism of superposition, while primitives realizing the

[1]`http://oreilly-qc.github.io`

[2]C211 Design Recipe anyone?

third step (*phase manipulation*) ensure that our results can actually be READ in a practical way.

These steps are usually implemented together and many times in iteration, in a manner dependent on the particular application. Rather than there being one all-purpose primitive for each step, we'll actually need an arsenal.

Each one of chapters 4-9 introduces one of the primitives[3] listed below:

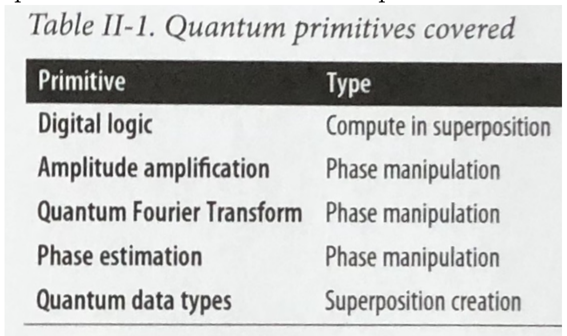


Table II-1. Quantum primitives covered

Primitive	Type
Digital logic	Compute in superposition
Amplitude amplification	Phase manipulation
Quantum Fourier Transform	Phase manipulation
Phase estimation	Phase manipulation
Quantum data types	Superposition creation

In each chapter we give a "hands-on" introduction to the primitive, then outline ways of *using* that primitive in practical applications. Finally, each chapter ends with a section entitled "Inside the QPU" giving more *intuition* into how the primitive works, often with a breakdown in terms of the fundamental QPU operations we've seen in Chapters 2 and 3.

The art of programming a QPU is to determine which combination of primitives from the table above works to construct a structure like the one shown on the previous page for a given application. In the last part of the book we show some examples of such constructions.

With some essential QPU primitives firmly in place, it's time to graduate from "How do QPUs work?" to "How can we use QPUs to do something useful?"

In Chapter 8, we will begin by demonstrating how useful data structures (rather than simple integers) can be represented and stored inside a QPU. Following this, in Chapter 9 we provide recipes to show how our arithmetic primitives from Chapter 4 can be used in a general class of applications that go under the name "quantum search". We then move on in Chapter 10 to presenting applications in computer graphics, before introducing Peter Shor's famous factoring algorithm in Chapter 11 and finally turning to applications of our QPU primitives to machine learning in Chapter 12.

The search for useful QPU applications is an ongoing one, the focus of intense research by thousands of specialists around the globe. Hopefully this part of the book will prepare and encourage you to take the first steps toward discovering QPU applications in areas that may not even yet have been explored.

[3] C211 Templates anyone?

Chapter 1

A Single Qubit

In this chapter we introduce a way to describe simple qubits, as well as a variety of QPU instructions to manipulate them. The random property of the READ operation is used to construct a quantum random number generator, and control over the relative phase in a qubit is used to perform[1] basic quantum cryptography[2].

The circle notation used to visualize the state of a qubit is also used extensively in the chapters ahead. In Chapter 2, we will extend the circle notation to deal with the behavior of multi-qubit systems, and introduce new QPU operations used to work with them.

[1]Run it at `http://oreilly-qc.github.io?p=2-4`

[2]For a more practical demonstration of the power in manipulating the relative phases of qubits, this chapter ends with a more complex program (there is also a Qiskit version of the program at the link above). The code presented uses the simple single-qubit QPU operations introduced in this chapter to perform a simplified version of Quantum Key Distribution (QKD). QKD is a protocol at the core of the field of quantum cryptography that allows the provably secure transmission of information. Suppose that two QPU programmers, Alice and Bob, are sending data to each other via a communication channel capable of transmitting qubits. Once in a while, they send a specially constructed "spy hunter" qubit, which they use to test whether their communication channel has been compromised. Any spy who tries to read one of these qubits has $\frac{1}{4}$ chance of getting caught. So even if Alice and Bob only use 50 of them in the whole transfer, the spy's chances of getting away are far less than one in a million. Alice and Bob can detect whether their key has been compromised by exchanging some conventional digital information, which does not need to be private or encrypted. After exchanging their messages, they test a few of their qubits by reading them out and checking that they agree in a certain expected way. If they disagree, then they know that someone was listening in.

Chapter 2

Multiple Qubits

As useful as single qubits can be, they're much more powerful (and intriguing) in groups. We've already seen in Chapter 1 how the distinctly quantum phenomenon of superposition introduces the new parameters of magnitude and relative phase for computation. When our QPU has access to more than one qubit, we can make use of a second powerful quantum phenomenon known as *entanglement*. Quantum entanglement is a very particular kind of interaction between qubits, and we'll see it in action within this chapter, utilizing it in complex and sophisticated ways.

But to explore the abilities of multiple qubits, we first need a way to visualize them. Can we extend our circle notation to multiple qubits? If our qubits didn't interact with one another, we could simply employ multiple versions of the representation we used for a single qubit. In other words, we could use a pair of circles for the $|0\rangle$ and $|1\rangle$ states of each qubit. Although this naive representation allows us to describe a superposition of any one individual qubit, there are superpositions of groups of qubits that it cannot represent.

As is explained in this chapter quantum relationship between multiple qubits demands that there must be a new kind of link between them to ensure a certain kind of logical consistency; this link is the new and powerful *entanglement* phenomenon.

Entangled multi-qubit states cannot be described in terms of individual descriptions of what the constituent qubits are doing, although you're welcome to try! This entanglement link is only describable in the configuration of the *whole* multi-qubit register. It also turns out to be impossible to produce entangled states from only *single-qubit* operations. To explore entanglement in more detail, we'll need to introduce multi-qubit operations.

Having seen how single- and multi-qubit operations allow us to manipulate the properties of superposition and entanglement within a QPU, we're ready to see how they allow us to *compute* with a QPU in new and powerful ways. In Chapter 4 we'll show how fundamental digital logic can be reimagined, but first in the next chapter we conduct a hands-on exploration of quantum teleportation. Not only is quantum teleportation a fundamental component of many quantum applications, but exploring it also consolidates everything we've covered so far about describing and manipulating qubits.

Chapter 3

Quantum Teleportation

In this chapter we introduce a QPU program allowing us to immediately teleport an object across a distance of 3.1 millimeters! The same code would work over interstellar distances, given the right equipment.

Although teleportation might conjure up images of magician's parlor tricks, we'll see that the kind of *quantum* teleportation we can perform with a QPU is equally impressive, yet far more practical—and is, in fact, an essential conceptual component of QPU programming.

For this example, rather than a simulator, we will use IBM's five-qubit actual QPU. You'll be able to paste the sample OpenQASM code provided into the IBM Q Experience website, click a button, and confirm that your teleportation was successful.

Teleportation is a surprisingly fundamental part of the operation of a QPU—even in straight computational applications that have no obvious "communication" aspect at all. It allows us to shuttle information between qubits while working around the "no-copying" constraint. In fact, most practical uses of teleportation are over very short distances within a QPU as an integral part of quantum applications. In the upcoming chapters, you'll see that most quantum operations for two or more qubits function by forming various types of entanglement. The use of such quantum links to perform computation can usually be seen as an application of the general concept of teleportation. Though we may not explicitly acknowledge teleportation in the algorithms and applications we cover, it plays a fundamental[1] role.

[1]Side-effects, monadic effects?

Chapter 4

Quantum Arithmetic and Logic

QPU applications often gain an advantage over their conventional counterparts by performing a large number of logical operations in superposition. A key aspect of this is the ability to apply simple arithmetic operations on a qubit register in superposition. In this chapter, we will look in detail at how to do this. Initially, we'll discuss arithmetic operations at the more abstracted level we're used to in conventional programming, dealing with integers and variables rather than qubits and operations. But toward the end of the chapter we'll also take a closer look at the logical operations making these up (akin to the elementary gates of digital logic).

The ability to perform digital logic in superposition is a core part of most QPU algorithms. In this chapter, we have taken a close look at ways to manipulate quantum data and even to perform conditional operations within superpositions. Performing digital logic in superposition is of limited use unless we can extract information from the resulting state in a useful way. Recall that if we try to READ a superposition of solutions to an arithmetic problem, we'll randomly obtain just one of them. In the next chapter, we will explore a QPU primitive allowing us to reliably extract output from quantum superpositions, known as amplitude amplification.

Chapter 5

Amplitude Amplification

This chapter introduces one of the core operations in many QPU applications.

By converting phase differences into magnitude differences, amplitude amplification allows a QPU program to provide useful output relating to phase information from a state that would otherwise remain invisible.

We explore the full power of this primitive in Chapters 10[1] and 12-13[2].

[1]Quantum Supersampling

[2]Shor's factoring algorithm

Chapter 6

QFT

In this chapter you have learned about one of the most powerful QPU primitives, the Quantum Fourier Transform (QFT). While the Amplitude Amplification primitive allowed us to extract information about discrete values encoded in the phases of our register, the QFT primitive enables us to extract information about patterns of information encoded in the QPU register. As we will see in Chapter 11, this primitive is at the core of some of the most powerful algorithms that we can run on a QPU, including Shor's algorithm, which first kick-started mainstream interest in quantum computing.

Chapter 7

Phase Estimation

In this chapter we explore a new QPU primitive, phase estimation. Phase estimation is also quite possibly the most challenging primitive thus far. This primitive uses three previously introduced concepts (phase kickback, construction of controlled unitaries, and the invQFT primitive) to achieve a great feat: it can extract information that QPU operations encode in the global phases of a register. It does so by transforming the global phase information into relative phase information in a second quantum register, and then applying invQFT to extract that information in a READ-able format. This operation will prove crucial for some of the machine-learning operations that we will encounter in Chapter 12

Chapter 8

Real Data

Fully fledged QPU applications are built to operate on genuine, unaccommodating data. Real data won't necessarily be as simple to represent as the basic integer inputs we've been satisfied with up until now. Thinking of how to represent more complex data within QPUs is thus well worth our effort, and a good data structure can be just as important as a good algorithm. This chapter sets out to answer two questions that we've previously sidestepped:

1. *How should we represent complicated data types in a QPU register?* A positive integer can be represented with simple binary encoding. What should we do with irrational, or even compound types of data, such as vectors or matrices? This question takes on more depth when considering that superposition and relative phase might allow entirely *quantum* ways of encoding such data types.

2. *How can we read stored data into a QPU register?* So far we've been initializing our input registers by hand, using WRITE operations to manually set a register's qubits to binary integers of interest. If we're ever to employ quantum applications on large swathes of data we'll need to read that data into QPU registers from memory. This is a nontrivial requirement as we may want to initialize a QPU register with a superposition of values something that conventional RAM isn't cut out for.

We'll start by addressing the first of these questions. As we describe QPU representations for increasingly complex types of data, we'll be led to introduce some truly quantum data structures and the concept of Quantum Random Access Memory (QRAM). QRAM is a crucial resource for many practical QPU applications. In coming chapters we rely heavily on the data structures introduced here. For example, the so-called amplitude encoding we introduce for vector data is at the heart of every quantum machine-learning application mentioned in Chapter 12.

Chapter 9

Quantum Search

In Chapter 5 we saw how the amplitude amplification (AA) primitive changes differences in phases within a register into detectable variations in magnitude. Recall that when introducing AA, we assumed that applications would provide a subroutine to flip the phases of values in our QPU register. As a simplistic example, we used the `flip` circuit as a placeholder, which simply flipped the phase of a single known register value. In this chapter we will look in detail at several techniques for flipping phases in a quantum state based on the result of nontrivial logic.

Quantum Search (QS) is a particular technique for modifying the `flip` subroutine such that AA allows us to reliably READ solutions from a QPU register for a certain class of problems. In other words, QS is really just an application of AA, formed by providing an all-important subroutine marking solutions to a certain class of problems in a register's phases.

The class of problems that QS allows us to solve is those that repeatedly evaluate a subroutine giving a yes/no answer. The yes/no answer of this subroutine is, generally, the output of a conventional boolean logic statement. One obvious problem that can be cast in this form is searching through a database for a specific value. We simply imagine a boolean function that returns a 1 if and only if an input is the database element we're searching for. This was in fact the prototypical use of Quantum Search, and is known, after its discoverer, as Grover's search algorithm. By applying the Quantum Search technique, Grover's search algorithm can find an element in a database using only $O(\sqrt{N})$ database queries, whereas conventionally $O(N)$ would be required. However, this assumes an unstructured database—a rare occurrence in reality—and faces substantial practical implementation obstacles.

Although Grover's search algorithm is the best known example of a Quantum Search application, there are many other applications that can use QS as a subroutine to speed up performance. These range from applications in artificial intelligence to software verification.

The piece of the puzzle that we're missing is just how Quantum Search allows us to find subroutines encoding the output of any boolean statement in a QPU register's phases (whether we use this for Grover's database search or other QS applications). Once we know how to do this, AA takes us the rest of the way. To see how we can build such subroutines, we'll need a sophisticated set of tools for manipulating QPU register phases a repertoire of techniques we term *phase logic*. In the rest of this chapter, we outline phase logic and show how QS can leverage it. At the end of the chapter, we summarize a general recipe for applying QS techniques to various conventional problems.

Chapter 10

Quantum Supersampling

From pixel-based adventure games to photorealistic movie effects, computer graphics has a history of being at the forefront of computing innovations. Quantum Image Processing (QIP) employs a QPU to enhance our image processing capabilities. Although very much in its infancy, QIP already offers exciting examples of how QPUs might impact the field of computer graphics.

This chapter demonstrates how exploring new combinations of QPU primitives with a little domain knowledge can potentially lead to new QPU applications. The ability to redistribute sampling noise also shows visually how sometimes we have to look beyond speedups to see the advantages of QPU applications.

It's also worth noting that the ability of Quantum Supersampling to produce novel noise redistributions potentially has relevance beyond computer graphics. Many other applications also make heavy use of Monte Carlo sampling, in fields such as artificial intelligence, computational fluid dynamics, and even finance.

To introduce Quantum Supersampling, we used a phase encoding representation of images in QPU registers. It is worth noting that Quantum Image Processing researchers have also proposed many other such representations. These include the so-called Qubit Lattice representation, Flexible Representation of Quantum Images (FRQI), Novel Enhanced Quantum Representation (NEQR), and Generalized Quantum Image Representation (GQIR). These representations have been used to explore other image processing applications including template matching, edge detection, image classification, and image translation to name but a few.

Chapter 11

Shor's Factoring Algorithm

If you'd heard about one application of quantum computing before you picked up this book, there's a good chance it was Shor's factoring algorithm. Quantum computing was mostly considered to be of academic, rather than practical, interest until Peter Shor's 1994 discovery that a sufficiently powerful quantum computer can find the prime factors of a number exponentially faster than any conventional machine. In this chapter, we take a hands-on look at one specific implementation of Shor's QPU factoring algorithm. Far from being a mere mathematical curiosity, the ability to quickly factorize large numbers can help break the Rivest-Shamir-Adleman (RSA) public-key cryptosystem.

This chapter presents a simplified version of what is typically a very complicated algorithm. In our version of Shor's algorithm, a few aspects were simplified for ease of illustration, although at the cost of generality. Without digging too deeply, the chapter ends by mentioning some of the necessary simplifications. More information can also be found in the online sample code.

Chapter 12

Quantum Machine Learning

At the time of writing, quantum machine learning (QML) is just about the greatest combination of buzzwords you could hope to synthesize. A lot is written about QML, and the topic is often (confusingly) both overhyped and undersold at the same time. In this section we'll try to give a flavor for how QPUs might transform machine learning, while also being careful to point out the caveats inherent in manipulating quantum data.

Useful QML applications require very large numbers of qubits. For this reason, our overview of QML applications is necessarily very high-level. Such a summary is also fitting given the rapidly changing nature of this nascent field. Although our discussion will be more schematic than pragmatic, it will heavily leverage our hands-on experience of primitives from earlier chapters. We summarize three different QML applications: solving systems of linear equations, Quantum Principal Component Analysis, and Quantum Support Vector Machines. These have been selected due to both their relevance to machine learning and their simplicity to discuss. These are also applications whose conventional counterparts are hopefully familiar to anyone who has dabbled in machine learning. We only give a brief description of the conventional progenitors of each QML application as it's introduced.

Quantum machine learning is still an extremely dynamic area of research. In this chapter three canonical examples are presented, but new developments in QML are constantly being made. At the same time, conventional approaches to machine learning are being inspired by QPU algorithms in fact, within the time period of us writing this book, all three of the QML applications presented in this chapter have inspired the development of conventional algorithms with similar runtime improvements. This should not shake your confidence in QML's potential; these results were unlikely to have been discovered without the inspiration of their QPU counterparts, showing that QML applications have far-reaching and unexpected consequences. There are also many other QML applications that we simply didn't have space to properly mention. These include efficient QPU applications for linear regression, unsupervised learning, Boltzmann machines, semidefinite programming, and quantum recommender systems (quantum recommender systems have also inspired improvements in conventional algorithms).

Chapter 13

Commencement

Before closing, we'll briefly introduce a few subjects we didn't have space to go into previously, and provide pointers on where to go to learn more about these and other topics in quantum computing. We won't go into too much depth here; ther aim is rather to link what you've learned so far to material reaching beyond the scope of the book.

Thus the following topics are adressed in this chapter: from circle notation to complex vectors; some subtleties and notes on terminology; measurement bias; gate decompositions and compilation; gate teleportation.

There's a "QPU Hall of Fame" section, another one on "Oracle-Based Algorithms" and one on "Quantum Programming Languages".

The chapter ends with a discussion on the promise of quantum simulation and a section on error correction and NISQ devices.

It is hoped that the book has provided the diligent reader with the conceptual tools and QPU intuition to further explore the fascinating topic of quantum computing. With the foundation you now have, the references we list in this section are good places to take your understanding of QPUs to the next stage. Be warned (but not dissuaded!) that these references often freely use the more advanced levels of linear algebra and other mathematics that we've aimed to avoid in this text.

There are interesting and useful lecture notes available online as well as many wonderfully comprehensive online resources from Aaronson[1], Preskill[2], Vazirani[3], Shor[4] and many[5] others[6].

[1]`https://www.scottaaronson.com/qclec/combined.pdf`
[2]`http://www.theory.caltech.edu/people/preskill/ph229/#lecture`
[3]`https://www.edx.org/course/quantum-mechanics-quantum-computation-uc-berkeleyx-cs-191x`
[4]`https://ocw.mit.edu/courses/mathematics/18-435j-quantum-computation-fall-2003/`
[5]`http://quantumalgorithmzoo.org/`
[6]`https://cosmosmagazine.com/physics/quantum-computing-for-the-qubit-curious`