Graduate Lecture Notes on Quantum Computing
From Digital Logic to Qiskit Simulation

Your Name

March 20, 2025

Contents

Chapter 1

Introduction

This document presents a rigorous treatment of quantum computing by bridging classical digital logic with modern quantum circuit simulation. We begin with the symbolic and algebraic formulation of the 1-bit half adder—a fundamental digital circuit—and extend these ideas into the quantum domain using Qiskit. Emphasis is placed on formal definitions, mathematical symbolism, and object-oriented programming concepts that are essential for both theoretical understanding and practical implementation.

Chapter 2

Classical Digital Logic: The 1-Bit Half Adder

2.1 Fundamentals of Digital Circuits

Digital circuits perform operations on binary variables. A *half adder* is one of the simplest circuits and is defined for two binary inputs $A, B \in \{0, 1\}$. It computes two outputs:

$$S = A \oplus B, \quad C = A \wedge B,$$

where $\oplus$ denotes the exclusive OR (XOR) and $\wedge$ denotes the logical AND.

Definition 2.1 (Half Adder)**.** *Let $A, B \in \{0, 1\}$. The* half adder *produces:*

- ***Sum:*** $S = A \oplus B$
- ***Carry:*** $C = A \wedge B$

This corresponds to the binary addition $1 + 1 = 10$, where the sum is 0 and the carry is 1.

2.2 Truth Table and Boolean Representation

The truth table is given by:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

An equivalent algebraic representation is:

$$S = A + B - 2AB, \quad C = AB,$$

or, in modulo-2 arithmetic:

$$S = (A + B) \mod 2.$$

Chapter 3

Foundations of Quantum Computing

3.1 Qubits and Quantum Gates

In contrast to classical bits, a *qubit* is described by a unit vector in a two-dimensional Hilbert space:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \text{with } |\alpha|^2 + |\beta|^2 = 1.$$

Quantum gates are represented by unitary matrices. For instance, the Pauli-X (NOT) gate is given by:

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

which acts as:

$$X\,|0\rangle = |1\rangle\,, \quad X\,|1\rangle = |0\rangle\,.$$

3.2 Measurement Postulate

Upon measurement in the computational basis, the probability of observing state $|i\rangle$ for a state $|\psi\rangle$ is:

$$p(i) = |\langle i \mid \psi\rangle|^2.$$

Theorem 3.1 (Measurement Postulate)**.** *Let $|\psi\rangle \in \mathcal{H}$ be a quantum state. Then the measurement in the standard basis yields the outcome $|i\rangle$ with probability*

$$p(i) = |\langle i \mid \psi\rangle|^2.$$

Chapter 4

Quantum Circuit Construction with Qiskit

4.1 Overview of Qiskit

Qiskit is a Python framework for creating, simulating, and executing quantum circuits. In Qiskit, quantum circuits are constructed as objects that encapsulate registers, gates, and measurements.

4.2 Basic Quantum Circuit Example

The following Python code initializes a quantum circuit and applies the Pauli-X gate:

```
from qiskit import QuantumCircuit

# Create a quantum circuit with 2 qubits.
c1 = QuantumCircuit(2)
print('Type␣of␣c1␣object:', type(c1))

# Apply the Pauli-X gate on the 0-th qubit.
c1.x(0)
print('Qubits␣in␣circuit␣c1:', c1.qubits)
```

Listing 4.1: Quantum Circuit Initialization in Qiskit

4.3 Mapping the Half Adder to a Quantum Circuit

To emulate the classical half adder in a quantum setting, one can define a function that constructs a sub-circuit representing the half adder. Formally, define a function:

$$f_{\text{HA}} : \{0,1\}^2 \to \mathcal{C}, \quad f_{\text{HA}}(A,B) = \mathcal{C}_{\text{HA}},$$

where $\mathcal{C}_{\text{HA}}$ is the corresponding quantum circuit.

```
def half_adder():
"""
Constructs a quantum circuit emulating a 1-bit half adder.
Returns a QuantumCircuit object with 2 qubits and 2 classical bits.
"""
qc = QuantumCircuit(2, 2)
# Implement the XOR operation with a CNOT gate.
qc.cx(0, 1)
# The AND operation may be emulated by further gate decomposition.
qc.ccx(0, 1, 0)  # This is illustrative; a proper AND gate might
   require ancilla.
return qc

# Integrate the half adder sub-circuit into a main circuit.
main_circuit = QuantumCircuit(4, 4)
ha_circuit = half_adder()
main_circuit.compose(ha_circuit, inplace=True)
```

Listing 4.2: Definition of half_adder() Function

Chapter 5

Object-Oriented Programming and Modular Design in Python

5.1 Classes and Objects

Python is an object-oriented programming language. A *class* is a blueprint for objects; for instance, the Qiskit class `QuantumCircuit` defines methods and attributes for circuit operations.

Definition 5.1 (Class and Object)**.** *A class $\mathcal{C}$ is defined as a tuple $(\mathcal{D}, \mathcal{F})$, where $\mathcal{D}$ represents member variables and $\mathcal{F}$ represents member functions. An object is an instance of a class.*

5.2 Variable Scoping and Naming Conventions

5.2.1 Variable Scope

- **Global Variables:** Defined outside of functions.
- **Local Variables:** Defined within functions.

If a local variable is not declared, the global variable of the same name is used.

5.2.2 Naming Conventions

Python naming follows:

- **Variables/Functions:** lowercase with underscores (e.g., `half_adder`).
- **User-Defined Classes:** CapitalizedCamelCase (e.g., `QuantumCircuit`).
- **Constants:** ALL_CAPS (e.g., `UN_TOUCHABLE`).

For example,

$$\texttt{Qubit(QuantumRegister(4, 'q'),0)} \equiv \texttt{qreg\_q[0]}.$$

5.3 Global vs. Member Functions

Global functions, such as `plot_distribution()`, operate independently of class instances. In contrast, member functions (methods) such as `QuantumCircuit.x()` operate on specific object instances.

Chapter 6

Quantum Circuit Simulation and Execution

6.1 Simulation and Measurement

After constructing a quantum circuit, simulation is performed using a quantum simulator. Measurements are applied to collapse the quantum state and extract classical outcomes.

Example 6.1 (Measurement Simulation). Assume a circuit is simulated and all qubits are measured. The probability distribution over the outcomes is given by the Born rule:

$$p(i) = |\langle i \mid \psi \rangle|^2.$$

Visualization of these outcomes is critical to verify the circuit's behavior.

6.2 Execution on IBM Quantum Hardware

To execute on real quantum hardware, one must authenticate with an IBM Quantum token:

$$\texttt{token} = \texttt{"YOUR\_TOKEN"}.$$

After job submission, the system queues the execution. The job list and retrieval of results are handled via the IBM Quantum Platform.

Chapter 7

Practical Exercises and Lab Instructions

7.1 Lab Assignment: Constructing a Quantum Half Adder

7.1.1 Task Overview

1. Implement the `half_adder()` function in Qiskit.
2. Append the returned sub-circuit to a main circuit.
3. Simulate the circuit and verify that all qubits are measured.

7.1.2 Expected Measurement Outcomes

For the given lab, the measurement values should correspond to:

`01, 01, 01, 10, 10, 11`

corresponding to the underlying binary additions:

$$\begin{aligned} \text{in1} + \text{in0} &\rightarrow \text{result11, result10} \quad (1 + 1 = 10), \\ \text{in3} + \text{in2} &\rightarrow \text{result13, result12} \quad (1 + 0 = 01), \\ \text{result10} + \text{result12} &\rightarrow \text{result21, result20} \quad (0 + 1 = 01), \\ \text{result11} + \text{result13} &\rightarrow \text{result23, result22} \quad (1 + 0 = 01). \end{aligned}$$

7.2 Guidelines for Google Colab

- **Uploading Notebooks:** Use the Google Colab interface to upload your `.ipynb` file.
- **Downloading Notebooks:** Download the file from Colab for submission.

Chapter 8

Overview of Lecture Slides

Below is a summary of the slide contents covered in these notes:

Slide 1: 1-bit half adder: digital circuit.

Slide 2: Binary arithmetic: $1 + 1 = 10$.

Slide 3: Quantum circuit representation using Python Qiskit.

Slide 4: Using Google Colab for Python notebooks.

Slide 5: Installation of the Qiskit package.

Slides 6–7: (Additional introductory content.)

Slides 8–12: Detailed discussion on classes and objects.

Slide 13: Function syntax and return values.

Slide 14: Python naming conventions.

Slides 15–16: (Supplementary topics.)

Slide 17: Modular programming: implementation as functions.

Slide 18: Local vs. global variables.

Slides 19–24: (Further elaboration on variable scope and circuit construction.)

Slide 25: Simulation of quantum circuits.

Slide 26: Advanced simulation techniques.

Slide 27: Visualization of measurement results.

Slide 28: Execution on IBM Quantum hardware.

Slide 29: IBM Quantum token management.

Slide 30: Results from IBM Quantum executions.

Slide 31: IBM Quantum Platform job list.

Slide 33: Retrieving results from quantum jobs.

Slide 34: Lab exercise instructions.

Slides 35–36: (Additional exercise guidelines.)

Slide 37: Further simulation details.

Slide 38: Measurement visualization techniques.

Slide 39: Uploading `.ipynb` files to Colab.

Slide 40: Downloading `.ipynb` files from Colab.

Appendix A

Further Reading and References

- Nielsen, M. A. and Chuang, I. L., *Quantum Computation and Quantum Information*.
- Cross, A. W., Bishop, L. S., Smolin, J. A. and Gambetta, J. M., *Open Quantum Assembly Language*.
- Official Qiskit documentation: `https://qiskit.org/documentation/`.