QUANTENBASIERTE SOFTWAREENTWICKLUNG

Bekannt und doch irgendwie anders

Für Quantencomputer zu programmieren erfordert spezielle Softwareplattformen – zum Beispiel die von Classiq.

Das israelische Unternehmen Classiq aus Tel Aviv [1] entwickelt Tools, mit denen sich Software für Quantencomputer erstellen lässt. Dieses Gebiet stellt sich für viele als böhmische Dörfer dar. Um sich dem Thema anzunähern, ist es sicherlich hilfreich, die Analogie zur klassischen Programmierung in C zu ziehen. So lässt sich erklären, wie Classiq eine neue Sprache und Plattform für dieses neue Computerparadigma entwickelt.

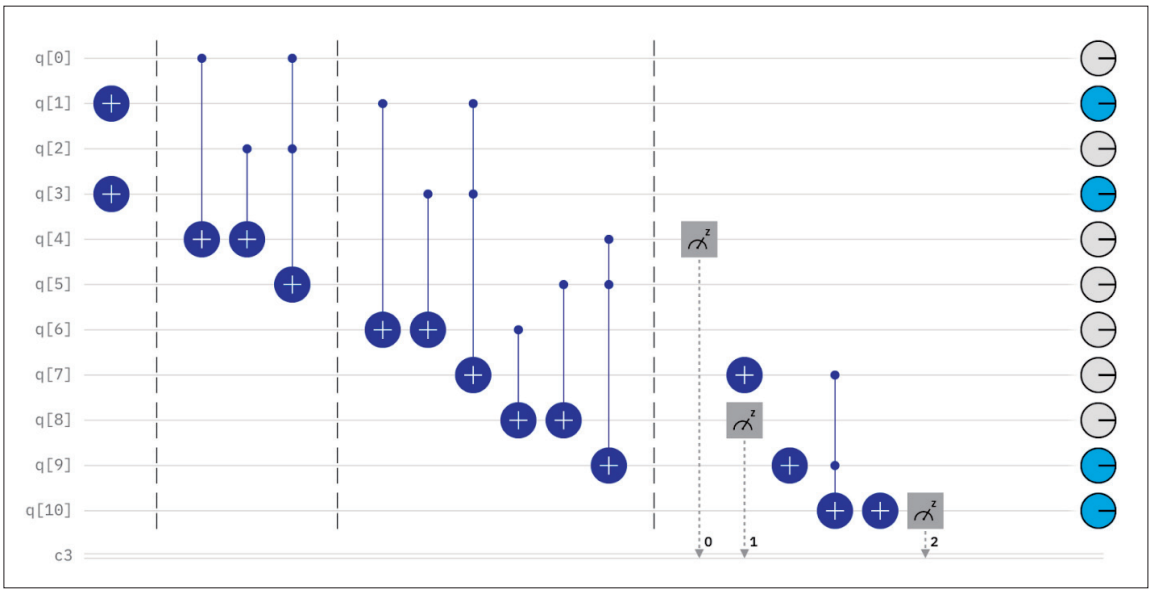
Quantencomputer basieren auf einem mathematischen Modell, das aus der Quantenphysik abgeleitet ist. Der Bau dieser völlig neuartigen Computer ist eine gewaltige technische Herausforderung, die bis vor nicht allzu langer Zeit noch als Fantasie galt. Doch in den vergangenen Jahren wurde die industrielle Nutzung von Quantencomputern dank der Durchbrüche von Computerriesen – wie IBM mit seinem 127-Qubit-Prozessor [2] und Googles Quantum Supremacy Experiment [3] – mehr und mehr zu einer Frage des Wann und nicht des Ob.

Die Motivation zum Bau von Quantencomputern liegt auf der Hand: Wenn es gelingt, einen solchen Computer zu bauen und die entsprechende Software dafür zu schreiben, könnten sich kritische Probleme lösen lassen, die klassische Computer nicht bewältigen können und nie bewältigen werden. So ließen sich beispielsweise neue Impfstoffe entwickeln, Versorgungskette optimieren, Moleküle in großem Maßstab simulieren, neue Algorithmen für maschinelles Lernen entdecken, Informationen besser schützen und vieles mehr.

Ein Blick zurück in die 1950er-Jahre zeigt einen langen Forschungs- und Entwicklungsprozess, der von den ersten Computerprototypen zu den heutigen modernen Prozessoren führte. Moderne Prozessoren zeichnen sich durch eine Befehlssatzarchitektur aus, welche die von ihnen verwendeten Befehle, Datentypen und Register festlegt. Bestimmte Befehle sind sowohl binär als auch in Assembler lesbar. Aber heute entwickelt niemand mehr große Softwareprojekte in Assembler, nicht einmal Linus Torvalds, der Vater von Linux.

In den 60er- und 70er-Jahren wurden Programmiersprachen wie Fortran und später C entwickelt, um eine Abstraktionsschicht über der Assemblersprache zu schaffen. Diese Sprachen revolutionierten die Softwareentwicklung und ermöglichten es Software-Ingenieuren, ihrer Kreativität freien Lauf zu lassen.

Bei einer Programmiersprache für Quantencomputer kann man sich eine Reihe von logischen Verknüpfungen vorstellen, die eine andere Semantik haben als die klassische Pro-



Ein Quantenschaltkreis, der das QUASM-Programm aus Listing 1 darstellt (Bild 1)

Listing 1: Mit Quantum Assembler 2 + 2 berechnen

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[11];
creg c[3];
// Diese einleitenden Zeilen initialisieren 11
// Quantenregister und 3 klassische Register.

x q[1]; // q[1]q[0] is binary 10 (decimal 2)
x q[3]; // q[3]q[2] is binary 10 (decimal 2)
// Alle Qubits beginnen mit einem Wert von 0. Oben
// werden die Werte des 2. und 4. Qubits auf 1
// geändert. Die ersten 2 Qubits werden für die erste
// Zahl und die zweiten 2 Qubits für die zweite Zahl
// verwendet.
barrier q; // Halbaddierer
cx q[0], q[4];
cx q[2], q[4]; // sum
ccx q[0], q[2], q[5]; // carry
// Ein Halbaddierer erhält 2 Eingänge und gibt dann
// die Summe und den Übertrag aus.
barrier q; // Volladdierer
cx q[1], q[6];
cx q[3], q[6]; // XOR1
ccx q[1], q[3], q[7]; // AND1
cx q[6], q[8];
cx q[5], q[8]; // XOR2
ccx q[4], q[5], q[9]; // AND2
barrier q; // OR
x q[7];
x q[9];
ccx q[7], q[9], q[10];
x q[10];
// Ein Volladdierer akzeptiert 3 Eingänge,
// einschließlich des vorherigen Übertrags. Er gibt
// die Summe und einen neuen Übertrag aus.
measure q[4] -> c[0];
measure q[8] -> c[1];
measure q[10] -> c[2];
// Das Ergebnis ist die Ausgabe in binärer Form. Das
// erste Qubit ist die Summe des Halbaddierers, das
// zweite Qubit ist die Summe des Volladdierers und
// das dritte der Übertrag des Volladdierers.
```

grammierung. Anstatt mit einer Folge von 0 und 1 zu arbeiten, arbeiten Quantengatter mit linearen Kombinationen dieser Folgen. Die Quantenprogrammierung verbindet Quantengatter miteinander und kann auch mit klassischen Logikgattern kombiniert werden.

Quantencomputer haben heute nicht mehr als etwa 100 Qubits (ein Qubit ist die Entsprechung eines Bit auf bisherigen Computern). Sie haben viele Beschränkungen und für Unternehmen wäre es eine Herausforderung, sie für komplexe Berechnungen einzusetzen. Die Programmierung dieser Quantencomputer erfolgt in Sprachen wie QASM (Quantum Assembly) oder Q#. In diesen Sprachen muss der Software-Ingenieur explizit angeben, welche Quantengatter verwendet werden müssen, und viele würden dies als noch näher an der Hardware als Assembler betrachten.

Listing 1 zum Beispiel enthält ein einfaches QASM-Programm, das den Wert von 2 + 2 berechnet [4][5]. Dies wird dann in einen Quantenschaltkreis übersetzt, der in Bild 1 zu sehen ist (erstellt von IBM Quantum Experience).

Wenn IBM im Jahr 2023 Computer mit 1000 Qubits zur Verfügung stellen wird [6], welche die Möglichkeit bieten, anspruchsvolleren Code zu erstellen, wird eine solche Low-Level-Programmierung zu einer Sisyphusarbeit, die fast schon unmöglich ist.

Um dieses Problem zu lösen, entwickelt Classiq die nächsten Komponenten des Quantensoftwaresystems.

Klassisches iteratives Programmieren

Wie sieht die Programmierung eines klassischen Computers heute aus? Das Ziel besteht natürlich in erster Linie darin, eine ausführbare Datei zu erstellen, egal mit welcher Programmiersprache. Bei C erstellt der Softwareentwickler grob gesagt *.c*- und *.h*-Dateien und kompiliert sie zu *.o*-Dateien. Diese werden dann mit bereits vorhandenen Bibliotheken (zum Beispiel libc) zu einer ausführbaren Datei verknüpft. Der Ingenieur kann den Code ausführen, wird aber wahrscheinlich auf Fehler stoßen. Um den Code zu debuggen, könnte der Ingenieur einen Debugger wie GDB oder statische Codeanalyse-Tools wie objdump oder IDA verwenden. Dies ist ein iterativer Prozess: Code schreiben, kompilieren, linken, ausführen und debuggen (Bild 2).

Inwiefern ist dies mit dem Schreiben von Quantensoftware vergleichbar? Die Classiq-Plattform bietet Ingenieuren und Entwicklern von Quantenalgorithmen eine funktionale und deklarative Schnittstelle. Sie müssen also erst einmal deklarieren, was die Designerlogik oder Funktionalität ist, die sie generieren möchten, und dann die Einschränkungen angeben, die der Code erfüllen muss – wie zum Beispiel das Nichtüberschreiten einer bestimmten Schaltungslänge, die Anzahl der verfügbaren Qubits, die gewünschte Genauigkeit und anderes mehr.

Um die Anwender zu unterstützen, bietet Classiq eine Bibliothek von Bausteinen wie Zustandsvorbereitung, Quantenarithmetik und maschinelle Lernoperationen. Diese Deklarationen werden in einem Modell (*.qmod*) gesammelt und an den Backend-Server von Classiq gesendet. Zusammen mit vordefinierten Bausteinen durchläuft dieses Modell einen Syntheseprozess: das Erstellen von Quantencode, der die Anforderungen erfüllt. Nutzer können das Ergebnis analysieren und prüfen, ob es die Spezifikationen erfüllt, oder es ▶

Listing 2: Der Grover-Search-Algorithmus

```
{
  "constraints": {
    "max_width": 25,
    "max_depth": 300
  },
  "logic_flow": [
    {
      "function": "GroverOperator",
      "function_params": {
        "oracle": {
          "expression":
            "(a  + 3 ^ b) % 4  == 3",
          "definitions": {
            "a": {"size": 3},
            "b": {"size": 3}
          }
        }
      },
      "add_as_single_gate": false
    }
  ],
  "preferences": {
    "draw_at_level": 1,
    "output_format": [
      "qasm"
    ]

  }
}
```

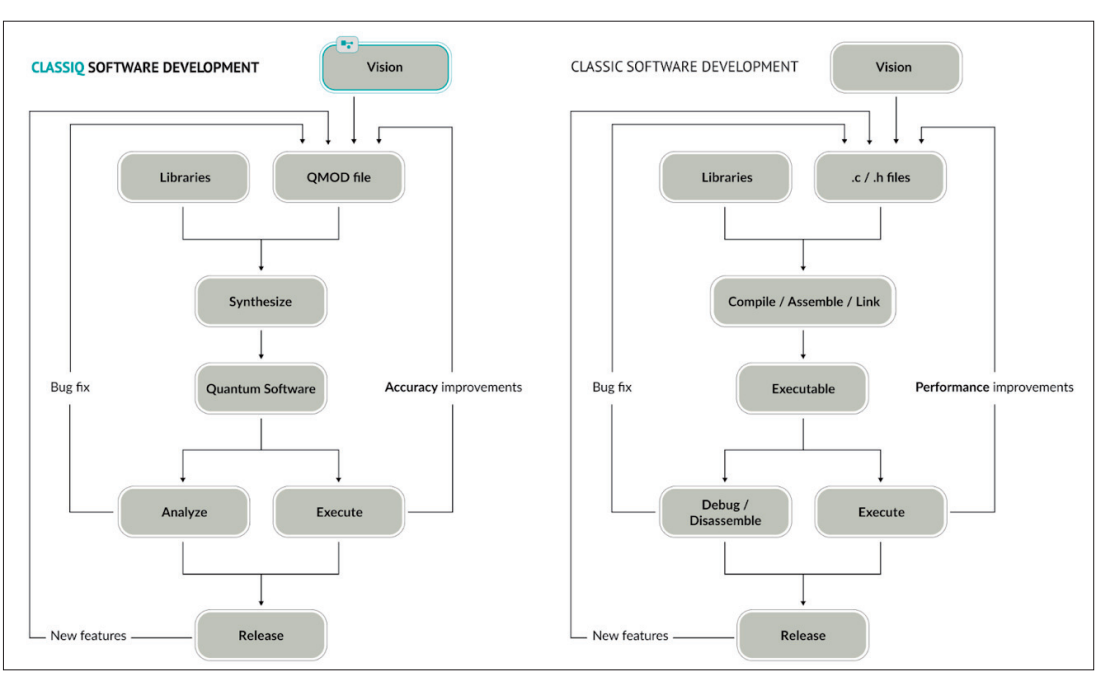
schnell auf einer Vielzahl von Quantencomputern oder Simulatoren ausführen. Im Rahmen dieses interaktiven und iterativen Prozesses lassen sich die Randbedingungen oder die Funktionsdefinition ändern, bis diese zum gewünschten Ergebnis führen. Wie bei der klassischen Kompilierung ist der ausgegebene Code schließlich viel besser als das, was manuell erreicht werden könnte.

Listing 2 zeigt zum Beispiel einen Classiq-Code, der einen berühmten Quantencomputer-Algorithmus namens Grover Search verwendet, um Werte für die Variablen *a* und *b* zu finden, welche die Berechnung *(a + 3 ^ b) % 4 == 3* erfüllen. Wie zu sehen ist, ist er wesentlich einfacher zu lesen. Der Code wird dann automatisch in eine Quantenschaltung umgewandelt, die wie in Bild 3 aussieht:

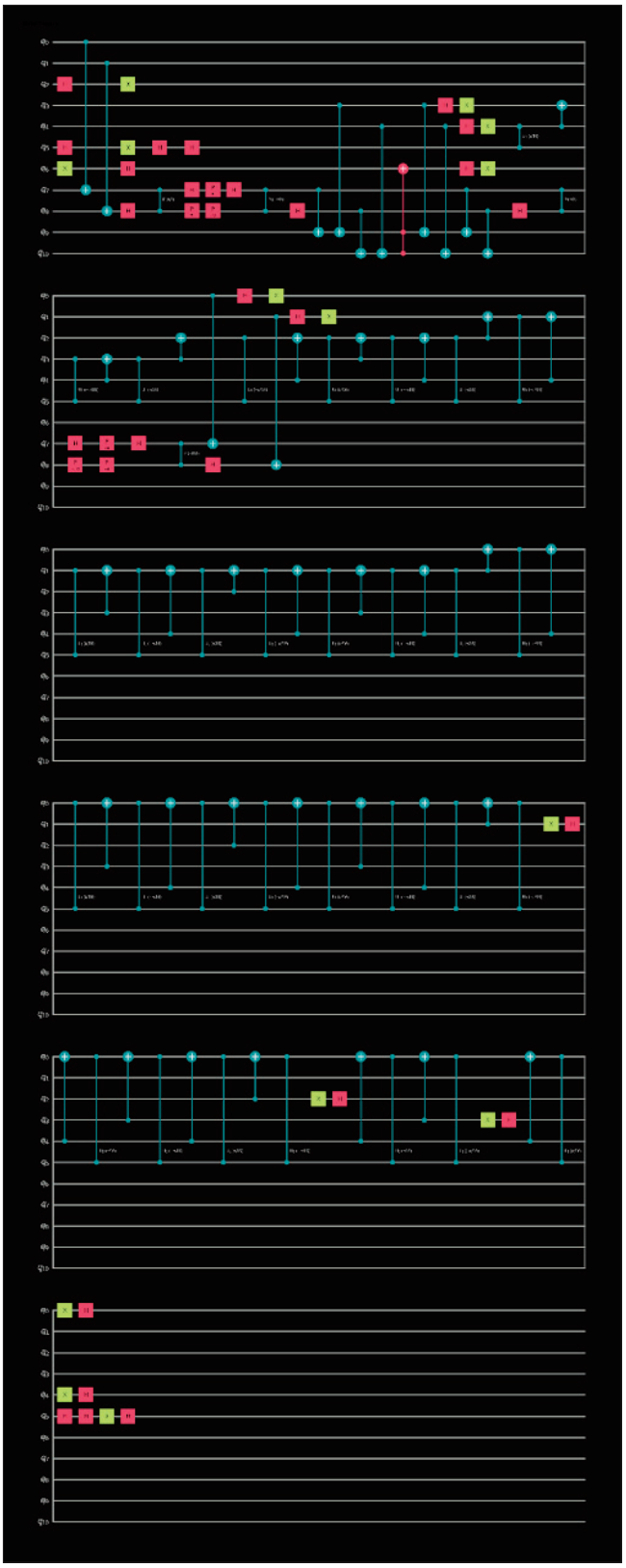
In diesem Fall wird die komplexe Aufgabe der Umwandlung des High-Level-Funktionsmodells in Quantengatter automatisch von der Plattform übernommen.

Trotz dieser Ähnlichkeiten gibt es doch einige profunde Unterschiede zwischen der klassischen und der Quantenprogrammierung:

- Die Quantenlogik ist komplexer als die klassische Logik, daher ist die Synthese von Quantenschaltungen sowohl im algorithmischen Entwurfsprozess als auch bei der Implementierung anspruchsvoller.
- Quantensoftware wird approximiert. Das bedeutet, dass ein Teil der Anforderungen der Grad der Genauigkeit des endgültigen Codes ist, der es erlaubt, die heutigen und zukünftigen Hardwarebeschränkungen zu berücksichtigen.



Der iterative Kreislauf der Software-Entwicklung (Bild 2)



Die Quantenschaltung zu (a + 3) ^ b % 4 == 3 (Bild 3)

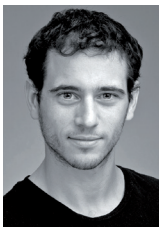
- Der Syntheseprozess erfolgt im Gegensatz zu einem lokalen klassischen Compiler über ein vernetztes API. Dies geschieht, um die Schnittstelle zu vereinfachen und die Vorteile eines SaaS-Modells zu nutzen (nahtlose Software-Upgrades, erweiterbare Kapazität und mehr).
- Die Fähigkeit, Quantencode zu debuggen, ist begrenzt, da der Prozess der Messung eines Qubits irreversibel ist. Daher ist der Prozess der Analyse und Validierung von Quantencode ein schwieriges algorithmisches Problem.
- Quantensoftware ist von Natur aus grafisch. Eine geeignete Entwurfsumgebung muss sowohl textuelle als auch grafische Methoden bereitstellen und auch den synthetisierten Code sowohl textuell als auch grafisch darstellen.

Die Welt der Quantensoftware befindet sich noch in den Kinderschuhen. Es gibt viele Parallelen zum klassischen Softwaredesign, aber der Weg ist größtenteils ungepflastert und erfordert intensive Forschung, Innovation und Einfallsreichtum. Einige der Herausforderungen, die Classiq in Angriff nimmt, sind:

- Das Erforschen und Umsetzen von Synthesealgorithmen, die einen Durchbruch bei Quantensoftware ermöglichen.
- Entwicklung einer reichhaltigen deklarativen Sprache, mit deren Hilfe reine und hybride (klassische / Quanten-)Algorithmen ausgedrückt werden können und die es Ingenieuren ermöglicht, ihrer Kreativität Ausdruck zu verleihen.
- Schaffen von Algorithmusbausteinen, die eine flexible Genauigkeit bieten und sich an die verfügbaren Quantenressourcen anpassen können.
- Werkzeuge für die Analyse und das Debugging von Quantencode innerhalb einer angemessenen Zeit.
- Eine skalierbare Infrastruktur für die nächsten Generationen von Quantencomputern.

Mir dem Bewältigen dieser Herausforderungen hofft Classiq, Werkzeuge bereitzustellen, um wichtige Forschungsaufgaben zu lösen, wie das Entwickeln neuer Impfstoffe, das Optimieren des Verkehrs, neue Arten von Batterien und Solarzellen, die Vorhersage von Wettermustern und vieles mehr. ■

[1] Classiq, https://de.classiq.io
[2] IBM Unveils Breakthrough 127-Qubit Quantum Processor, www.dotnetpro.de/SL2210Quanten1
[3] Hello quantum world! Google publishes landmark quantum supremacy claim, www.dotnetpro.de/SL2210Quanten2
[4] A.N.A.K.I.N., Quantum Full Adder, www.dotnetpro.de/SL2210Quanten3
[5] Quantum-Full-Adder bei GitHub, www.dotnetpro.de/SL2210Quanten4
[6] IBM, IBM's roadmap for scaling quantum technology, www.dotnetpro.de/SL2210Quanten5



Ofek Kirzner
ist Vizepräsident für Forschung und Entwicklung beim Quantensoftware-Unternehmen Classiq. Mit der Forschung für seine Masterarbeit gewann er den Internet Defense Prize 2021 von Facebook und USENIX. Kirzner ist Absolvent des renommierten „Talpiot"-Programms.

dnpCode A2210Quanten