TECHNICAL ARTICLE

Open Quantum Assembly Language

Author: Afrasiyab Khan

October 22, 2022

***Abstract*—Quantum Computing is one of the most exciting inventions that is still in its infancy. Also called the "future of Computers", Quantum Computing could change the world forever. Some even attribute Quantum Computing with being the "next Industrial Revolution". This is a technical article on the Open Quantum Assembly Language, also known as OpenQASM (pronounced open-qazm). The article will introduce the technology, discuss features & skepticism, go over its uses, and discuss its past and future.**

I. Introduction to Open Quantum Assembly Language

Open Quantum Assembly Language is IBM's implementation of the Quantum Assembly Language. It has now become the de facto standard in the Quantum Computing field, with OpenQASM 3, the newest iteration, paving the way. OpenQASM is used to program quantum circuits. In simple words, it is the "Assembly Language" for quantum computers, just like the one for traditional computers. IBM's implementation, OpenQASM, is a nice mixture of Assembly Language and the C programming language, geared for the programming of quantum computers.

OpenQASM makes it significantly easier to program quantum circuits. The language is translated & compiled into the basic fundamental operations for quantum computers (what you would call Machine Language in terms of traditional computers). These fundamental operations are what perform the task, much like traditional computers. However, all the operations are strictly in the quantum-realm of computing. [1]

A brief History

OpenQASM was first proposed in 2017 as an alternative and a major upgrade to over the existing quantum Programming technologies. It made programming quantum circuits using high level languages and tools possible. Moreover, the 2017 paper by a team of quantum physicists was specifically designed for the IBM Q Experience, which, back then, was IBM's quantum computing platform. Thanks to OpenQASM and other technological advancements, IBM brought its quantum computing platform on the cloud. [2] For quite some time, quantum programming lacked a low-level programming language. There was Q, and frameworks like QDK, but not a true low-level language for quantum computation. OpenQASM solved this problem and filled the gap, with Qiskit as its framework.

One of the programming interfaces for the IBM Quantum Services is the open quantum assembly language (OpenQASM 2), which was proposed as an imperative programming language for quantum processing based on earlier versions of QASM. Since OpenQASM 2 was released, it has kind of established itself as a standard, allowing numerous independent tools to use it as its standard interchange format. Additionally, OpenQASM2 played a vital role in in quantum computing research and technology. Traditional descriptions of quantum computation in these machine-independent languages follow the quantum circuit paradigm. [2] Currently, at the time of writing this, OpenQASM 2 is the current standard. However, OpenQASM 3 is currently in experimental phase and it aims to be better than its predecessor in many ways. These include extended operations beyond gate-level quantum circuits. This functionality solves many hurdles faced by OpenQASM 2, as it is, although powerful, but not very expressive and sometimes limits the user. [3] Here are some of the advantages OpenQASM 3 will have over OpenQASM 2:

- OpenQASM 3 will be a more complete language with classical programming support.
- Backward compatibility with OpenQASM 2 applications, so it will be easy or near seamless to move fro OpenQASM 2 to 3.
- Extended quantum circuits and computational routines.
- Instruction semantics for gate-scheduling.
- Ability to implement more versatile circuits over a vast range of use-cases.
- Better parser performance.

II. Importance of OpenQASM

Before we discuss the technical aspects and more about OpenQASM, we must understand what exactly does OpenQASM brings to the table and how it effected the quantum computing world.

- **Quantum train of thought:** OpenQASM is classified as a low-level language, much like Assembly language for classical computers. Just like ASM forces the programmer to think at low-level, OpenQASM does the same, as it's primary functionality is built on quantum bits (qubits) and gates. The programmer must think "quantumly" to work in OpenQASM. [2]
- **Broad use-cases:** A quantum circuit, or any similar representation of one, can be expressed using the OpenQASM 3 language as a collection of (quantum) basic blocks and flow control instructions. This is distinct from a high-level language's function and is faster. Programs that produce families of quantum circuits may be described in high-level languages. They might have features to specify how to turn classical reversible programs into quantum implementations, or they might have garbage collection and memory management systems for quantum computations.
- **Optimizations:** High-level optimizations (passes) with families of quantum operations whose precise parameters may not yet be known. At this level, these passes might be used just once and optimize any program instance that is run subsequently. This allows for a high-degree of optimization over high-level implementations, spanning multiple circuits. [3]
- **Built like a high-level language:** Many users would prefer to manually construct straightforward quantum programs using a domain-specific language that is expressive, despite the fact that OpenQASM is not a high-level programming language. High-level information in intermediate representations is often required by researchers/developers that study circuit compilation in order to guide the optimization and algorithmic synthesis.

Although developing programs at a reasonably high level is more convenient for a lot of such use-cases, timing or pulse-level gate descriptions are frequently needed to be manually changed throughout the program. This is where OpenQASM 3 shines. [3] Moreover, given the hardware limitations, hardware engineers that create traditional controllers and waveform generators choose languages that are easy to build. OpenQASM 3 is geared to take into account all of these prospective use-cases.

III. Technological Details - How OpenQASM works

OpenQASM is specifically designed to work on a low-level, according to its developers.

> "It is not our intent; however, to transform OpenQASM into a general-purpose programming language suitable for classical programming." [3]

Classical calculations that must communicate with the quantum hardware are an integral part of a generic quantum application. For instance, certain applications need pre-processing of the data, such as in Shor's algorithm, or post-processing to calculate Pauli operator expectation values. Pre-processing is also required to create additional circuits in the outer loop of several variation-based algorithms. Thus, for such applications, classical languages like Q are too slow to execute within the coherence time of the quantum hardware. OpenQASM executes in a near-time environment, which is easier to define in current classical programming frameworks. OpenQASM is focused on the real-time domain which works tightly with the quantum computer hardware.

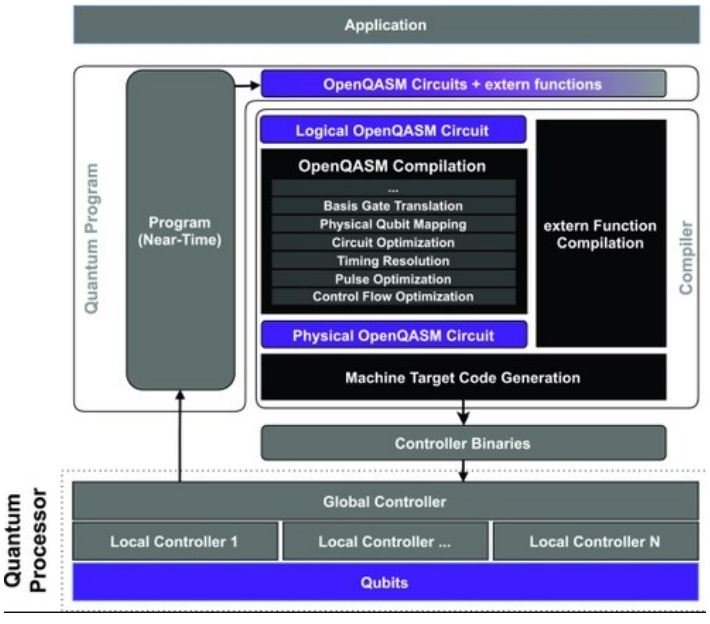


Fig. 1. How OpenQASM works. [1]

The above figure illustrates the Compilation-Execution model of OpenQASM. The program is Written in terms of a quantum program. A host computer near the quantum core will run certain parts of the quantum program (that are purely classical) during runtime.

OpenQASM as a language: A deep dive

Every OpenQASM program starts with the versionheader, which tells the transpiler what version of OpenQASM the program is typed in. The flow of a quantum program's compilation and its execution (as seen in Fig. 1), is written in terms of a quantum program or an application. Some areas of a quantum program can be written in pure classical programming languages (like Q) and executed in a near-time setting. Payloads are released by the quantum program to be executed on a QPU (Quantum Processing Unit). Extended quantum circuitry and external real-time classical functions make up this payload. The language used to describe quantum circuits is OpenQASM, and it supports interface calls to external classical functions. The circuit payload may include higher-order elements that are optimised before OpenQASM is created. This allows for a level of abstraction between OpenQASM and the interfaces.

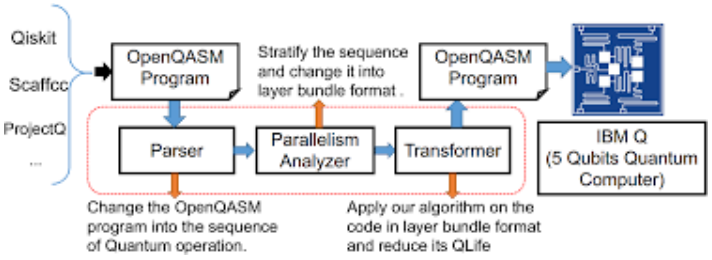


Fig. 2.

The qubit mapping, the gates used, the timing & pulses, and the control flow of the circuits defined with the IR can all be transformed and optimised to a high degree by an OpenQASM compiler. A target code generator creates the binaries for the QPU using the final physical circuit, while utilizing external functions.

```
1 OPENQASM 2.0;
2 include "qelib1.inc";
3 qreg q[2];
4 creg c[2];
5 h q[0];
6 cx q[0],q[1];
7 measure q[0] -> c[0];
8 measure q[1] -> c[1];
```

Fig. 3. A small OpenQASM program.

Similar to most high-level languages, comments in OpenQASM are typed as:

```
// This is a single-line comment.
/* This is a
    multi-line comment. */
```

Individual qubits and classical bits are represented by the qubit and bit types, respectively, while n-bit quantum and conventional registers are represented by the qreg[n] and creg[n] types, respectively. OpenQASM 2.0 has only four variable types total.

```
    bit classical_bit;
    qubit quantum_bit;
    qreg quantum_register[];
    creg classical_register[];
```

With a few additions, OpenQASM 3.0's basic syntax is relatively similar to that of OpenQASM 2.0. Several new types, including int, uint, float, and bool, are included in OpenQASM 3.0. OpenQASM incorporates several special features for dynamic quantum circuits while drawing influence from type systems seen in conventional programming languages. In OpenQASM 3, for-loops and while-loops are also introduced. By allowing the inclusion of an else block and several instructions inside the if statement's body, it also increases the if statement's scope.

```
    int signed_integer[8] =-42;
    uint unsigned_integer[8] = 42;
    float floating_point_number[8] = 0.42;
    bool boolean = true;
```

With these new types, OpenQASM 3 allows for more dynamic quantum circuits to be programmed. This ability is enhanced ten folds by the introduction of loops in OpenQASM 3.0:

```
    uint count[4] = 0;
    while (count < 10) {
        count++;
    }
```

Moreover, OpenQASM also supports conditional statements with high-level-like syntax.

```
    //Example 1
    if(c_reg==int) <Quantum Operation>

    //Example 2
    if(c_reg==1) U(pi/2, 0, pi) q_reg[0];

    //Example 3
    if(c_reg==4) CX q_reg[0], q_reg[1];
```

This can be further optimized by using delays and stretches.

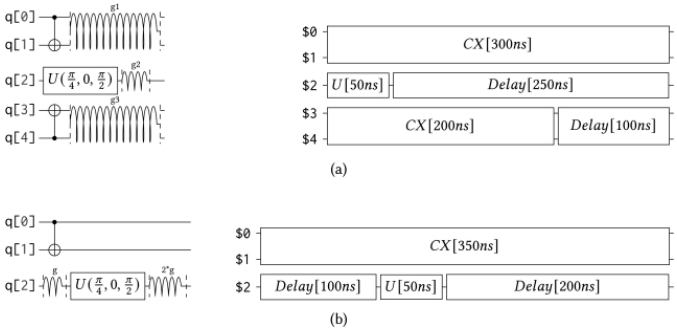


Fig. 4. Delay feature in OpenQASM. [3]

As we can see from the syntax, OpenQASM is heavily inspired by the C programming language. The syntax is a lot similar, which makes OpenQASM relatively easier to program as compared to something like ASM or MASM.

IV. Discussion - Problems with OpenQASM

Both OpenQASM versions have some obvious problems and grey areas. We'll discuss a few of the main ones:

No Manual Memory Allocation

To start with, OpenQASM does not explicitly describe manual memory allocation. This leads one to believe that memory allocation in OpenQASM must be automatic. This is very unusual for a low-level programming language, as manual memory allocation is crucial for performance-intensive programs. Even C, which is a high-level programming language, offers manual memory allocation. [3]

No Garbage Collection

Neither the official documentation nor white papers make reference to garbage collection. Given that most quantum computers only have a small amount of memory (less than 120 qubits), garbage collection is very crucial, so it is perplexing that OpenQASM's developers chose inefficient automatic memory allocation rather than manual memory allocation, which would enable programmers to make better use of a quantum computer's constrained resources.

Memory Deallocation/Freeing up Memory

Even with OpenQASM 3, there is still no method to delete variables or access the hardware qubits that correspond to a program's variables like there is in C. However, the reset keyword may be used as a pseudo memory deallocator, as it clears bits/registers.

Ambiguous Memory Usage

Since OpenQASM doesn't have a pointer type, its memory usage is opaque. OpenQASM professes to be an "assembly language," yet its approach to memory management demonstrates that it is more high-level than even C, which it is inspired by. This leads to the more fundamental issue that OpenQASM's design philosophy is rather flawed.

REGISTERS, GATES & THEIR NOTATIONS

In OpenQASM, registers and qubits are treated in a rather similar way. This results in a lot of problematic syntactical issues with the gate notations. Implementations of complex, scalable algorithms are hard and syntactically demanding, and require a bunch of arguments to be passed to the gate. This makes implementing complex quantum circuits unnecessarily difficult.

However, despite the shortcomings, OpenQASM 3 makes up for everything with its abilities and performance. It is a very capable language. The developers, according to the documentation, have considered adding support for resource and memory management, however, it was not adopted:

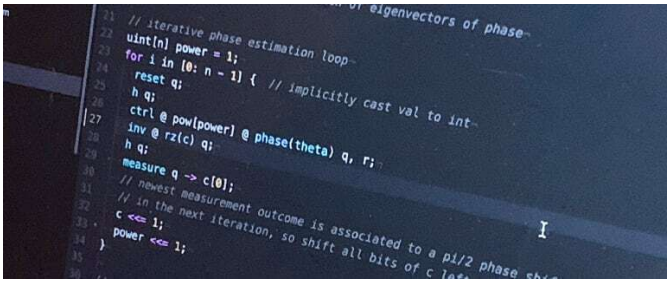
> "Some languages such as Q and ProjectQ support resource and memory management, in the form of mid-circuit allocation and de-allocation of qubits. Some languages such as Scaffold allow for the expression of classical code that will then be compiled to reversible quantum circuits (e.g., oracles). In each of these cases, OpenQASM intentionally remains lower-level and more grounded in executable circuits, aim at keeping the compiler's complexity more manageable. [3]"

CONTROL FLOW

Unstructured control flow is present in some languages, such as Quil, in the form of explicit program labels and JUMP/TO statements, which are more akin to the syntax of a traditional assembly language. However, OpenQASM only has a linear control flow. This might be a disadvantage in some use-cases, where another low level quantum language like Quil, shines.

The developers have addressed this design choice, citing the reason for this to be backwards compatibility between OpenQASM 2 and OpenQASM 3. Thus, OpenQASM 3 works from the first instruction, in a global scope, maintaining the classical control flow throughout. [3]

V. CONCLUSION



With a particular focus on their physical implementation and the interplay between classical and quantum computing, OpenQASM 3 has introduced language features that both broaden and deepen the breadth of quantum circuits that can be implemented. OpenQASM 3 has an unparalleled look and feel and is undoubtedly the most consistent low-level quantum programming language. Even though some aspects of it are non-ideal, OpenQASM 3 is a very powerful language that is still heavily under development. The developers will fix its shortcomings as advancements are made.

With a relatively easy syntax and a passionate team of developers behind it, OpenQASM is headed for success in the right direction. The developers are already hard-at-work considering, testing and implementing new features.

> "We fully expect the language to continue to evolve over time driven by real-world usage and hardware development. In particular, there are already proposals under consideration to modify implicit type conversions or enable code re-use through generic functions. [3]"

Moreover, OpenQASM is one of the best low-level programming languages for quantum computation and circuits. With OpenQASM 3, it offers unparalleled features, performance and scope. With constant developments and research, OpenQASM is going to maintain its status as the de facto standard of the industry, as a low-level programming language, just like Assembly for traditional computers was back in the day.

The development of OpenQASM and quantum computing in general is still in its infancy. The Qiskit community recently introduced a technical steering committee for OpenQASM. As we find the best techniques to operate and program quantum computers to realise the promised benefit of these machines, we anticipate that the language will evolve further in a good way. [4]

The contributions to OpenQASM that are focused on negotiated solutions, the formation of the technical steering committee will offer transparent and simple approaches. These procedures make sure that the entire community can influence OpenQASM's future, providing people and organisations peace of mind that their contributions will be valued and taken into account. [4]

REFERENCES

[1] [1] Zurek, Wojciech (2002). "Decoherence and the Transition from Quantum to Classical—Revisited" Los Alamos Science. 27.

[2] [2] [12]A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open Quantum Assembly Language," Open Quantum Assembly Language, Jul. 2017. https://arxiv.org/abs/1707.03429

[3] [3[1]A. Cross, T. Alexander, S. Heidel, J. M. Gambetta, and B. R. Johnson, "ACM Digital Library," ACM Digital Library, Sep. 2022.

[4] [3]Qiskit, "Introducing a Technical Steering Committee for OpenQASM — by Qiskit — Qiskit — Medium," Medium, Sep. 15, 2021. https://medium.com/qiskit/introducing-a-technical-steering-committee-for-openqasm3-f9db808108e1