

Qiskit

2021

**QAMP  
Checkpoint  
#2**

#23



# Status Update Report

---

## **Overview - OpenQASM 3.0 Reference Implementation**

Mentor: Jack Woehr [@jwoehr](#)

Team: Abeer Vaishnav [@AbeerVaishnav13](#) | Adrien Suau [@nelimee](#) | Vishal Bajpe [@mrvee-qC](#)

Github Repo: <https://github.com/mentor-fall2021-openqasm>

Website: <https://mentor-fall2021-openqasm.github.io>

## **Progress Summary.**

The version 1 of the implementation showcased on October 7 - Checkpoint #1 now has several fixes and changes with regards to the translator functionality with conformance to the OpenQASM 3.0 specification. Here are a few changes summarized on the [v1 implementation](#):

- Type system flaws identified and fixed in conformance to specification
- Implementations for quantum and classical registers now robust in aliasing.
- Initial implementation of 'duration' as per specification
- Cleanup and reconciling minor changes that further strengthen stricter conformance to implementation according to specification.

## **Contribution Summary:**

- [PR #295](#) : Suggestion to add Scope for power operator for complex types in language specification. Discussions taken up at TSC meeting. Discussions of requirement of dot notation of complex numbers for few rare use case implementations.
- [Issue #296](#) : Following discussion with [@taalexander](#) at IEEE Quantum Week on OpenPulse grammar implementation and Jacks further discussions, team is now added to the openpulse closed group for discussion with guidance from people and teams working on OpenPulse.
- [PR #269](#) : PR merged this month. Team contributed to bug and implementation fixes for merged reference AST implementation
- [Issue #304](#) : Bug report for openqasm3 pyPI package
- Presented progress and project overview at Qiskit advocate session at IEEE Quantum Week 2021

# Goals and further direction

---

## Goals and further direction:

- Reconstruct and restart current repository to depend on openqasm package rather than forks of openqasm repository and outsource package dependencies for openasm parser, ast and other components
- Work in support of PR #295 and fulfillment of Issue #296.
- Take inspirations from discussions in Issue #296 and start writing to invent own grammar and connect it to openpulse, eventually towards an implementation for translating to qiskit-pulse/ qiskit-pulse builder syntax.
- Take opinions and hints from openpulse invested parties in the discussion in Issue #296 while creating it along. Have received guidance support from openpulse closed group

1

Implement new discretized structure for repository

2

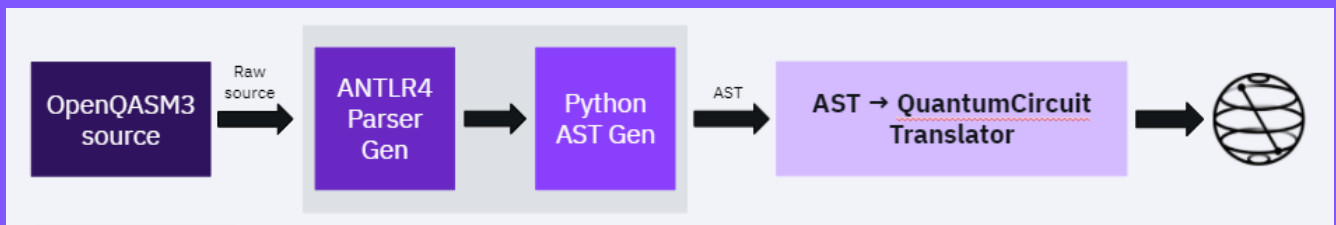
Formulation of own version of grammar for openpulse implementation

3

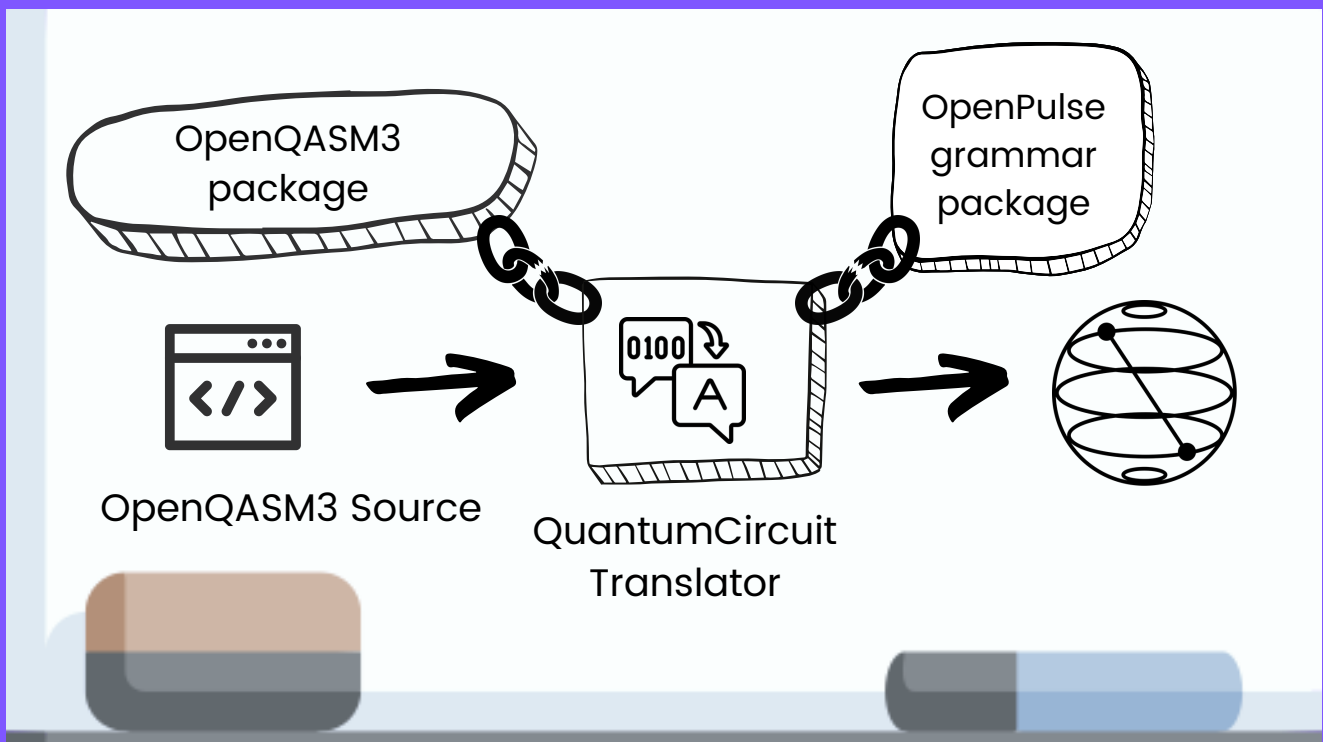
Discuss and take guidance from openpulse team on grammar formulation

# Current architecture vs proposed architecture

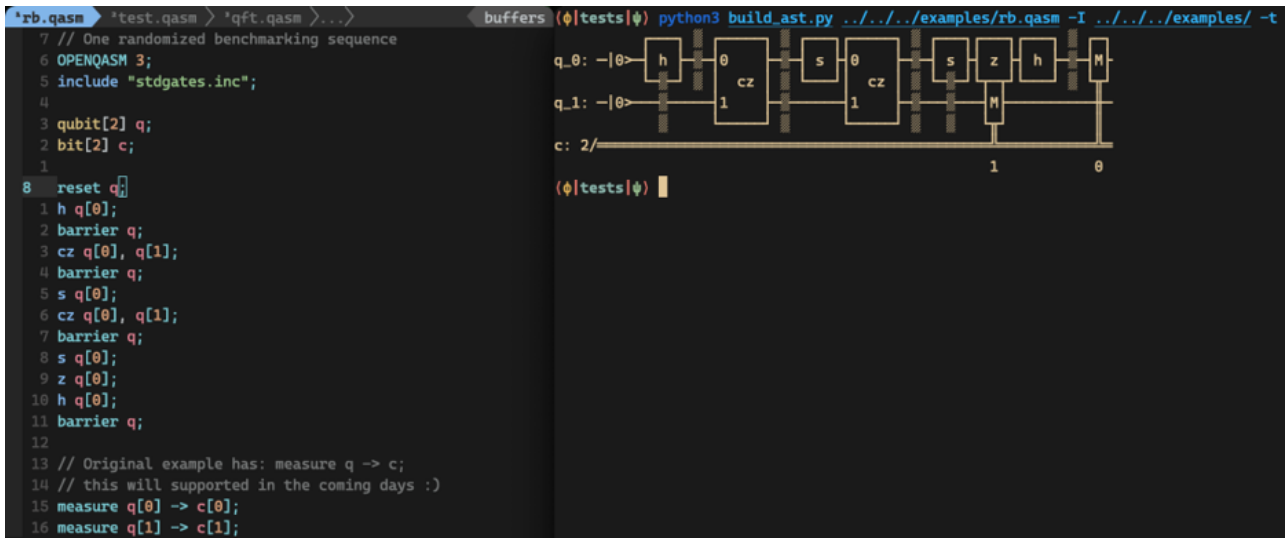
## Current



## Proposed



# Randomized Benchmarking sample



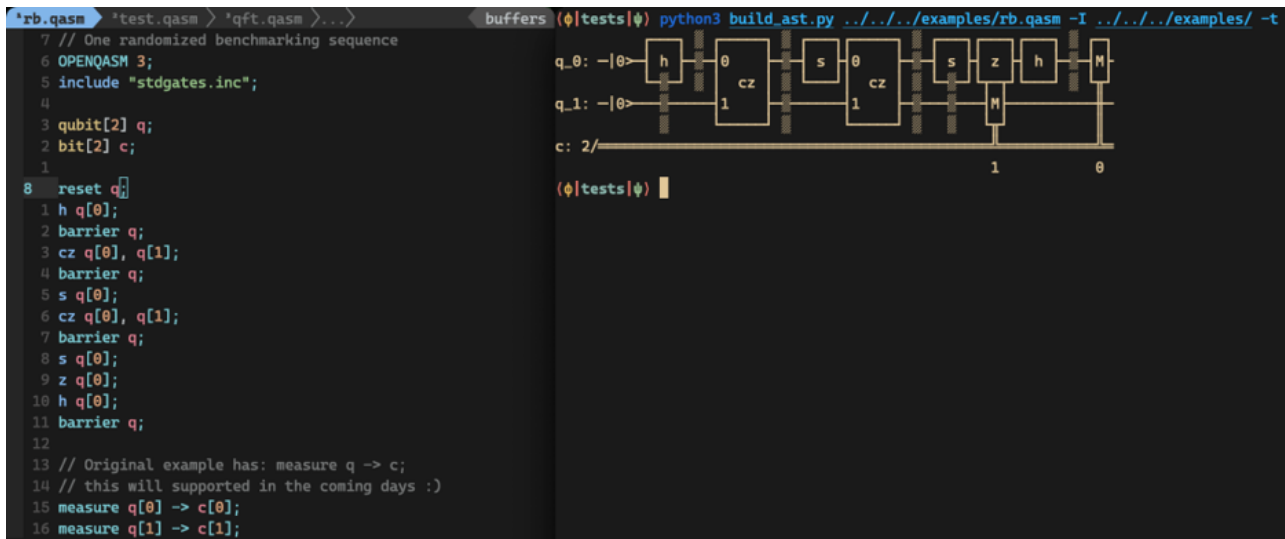
The screenshot displays a terminal window with the following QASM code on the left:

```
*rb.qasm > 'test.qasm > 'qft.qasm > ...> buffers (phi|tests|psi) python3 build_ast.py ../../examples/rb.qasm -I ../../examples/ -t
7 // One randomized benchmarking sequence
6 OPENQASM 3;
5 include "stdgates.inc";
4
3 qubit[2] q;
2 bit[2] c;
1
8 reset q;
1 h q[0];
2 barrier q;
3 cz q[0], q[1];
4 barrier q;
5 s q[0];
6 cz q[0], q[1];
7 barrier q;
8 s q[0];
9 z q[0];
10 h q[0];
11 barrier q;
12
13 // Original example has: measure q -> c;
14 // this will supported in the coming days :)
15 measure q[0] -> c[0];
16 measure q[1] -> c[1];
```

On the right, a quantum circuit diagram is shown for two qubits, q.0 and q.1, both initialized to  $|0\rangle$ . The circuit consists of the following operations:

- Qubit q.0: Hadamard (h) gate, followed by a  $\theta$  rotation gate, a CNOT (cz) gate with q.1 as control, an S gate, another  $\theta$  rotation gate, a second CNOT (cz) gate with q.1 as control, an S gate, a Z gate, and a final Hadamard (h) gate.
- Qubit q.1: A CNOT (cz) gate with q.0 as control, followed by a measurement (M) gate.
- Classical control: A CNOT (c) gate with q.1 as control and target c[0]. A second CNOT (c) gate with q.0 as control and target c[1].

# Custom gate and loop sample



The screenshot displays a terminal window with the following QASM code on the left:

```
*rb.qasm > 'test.qasm > 'qft.qasm > ...> buffers (phi|tests|psi) python3 build_ast.py ../../examples/rb.qasm -I ../../examples/ -t
7 // One randomized benchmarking sequence
6 OPENQASM 3;
5 include "stdgates.inc";
4
3 qubit[2] q;
2 bit[2] c;
1
8 reset q;
1 h q[0];
2 barrier q;
3 cz q[0], q[1];
4 barrier q;
5 s q[0];
6 cz q[0], q[1];
7 barrier q;
8 s q[0];
9 z q[0];
10 h q[0];
11 barrier q;
12
13 // Original example has: measure q -> c;
14 // this will supported in the coming days :)
15 measure q[0] -> c[0];
16 measure q[1] -> c[1];
```

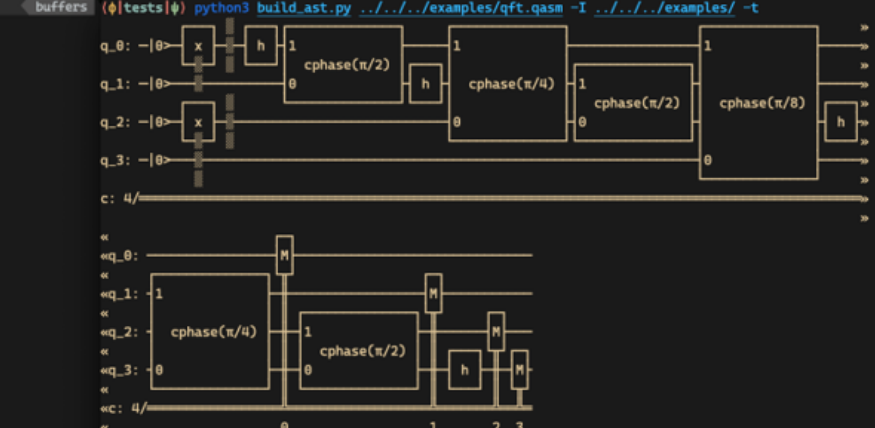
On the right, a quantum circuit diagram is shown for two qubits, q.0 and q.1, both initialized to  $|0\rangle$ . The circuit consists of the following operations:

- Qubit q.0: Hadamard (h) gate, followed by a  $\theta$  rotation gate, a CNOT (cz) gate with q.1 as control, an S gate, another  $\theta$  rotation gate, a second CNOT (cz) gate with q.1 as control, an S gate, a Z gate, and a final Hadamard (h) gate.
- Qubit q.1: A CNOT (cz) gate with q.0 as control, followed by a measurement (M) gate.
- Classical control: A CNOT (c) gate with q.1 as control and target c[0]. A second CNOT (c) gate with q.0 as control and target c[1].

# Demo Screenshots

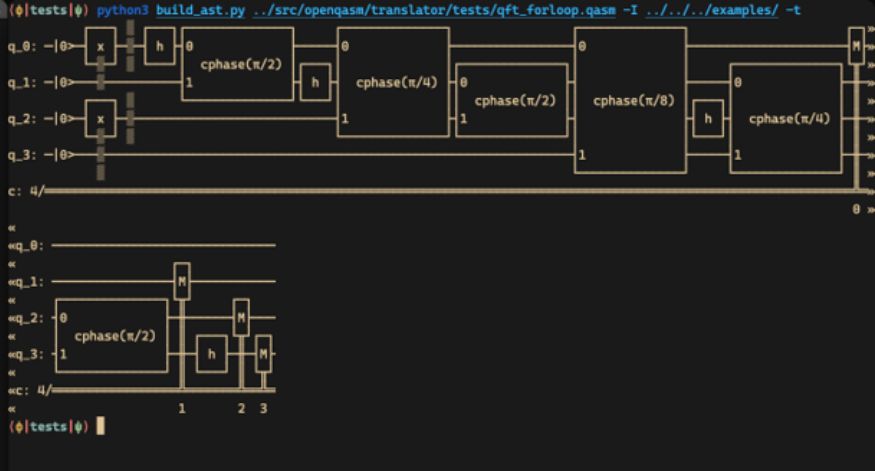
# QFT | OpenQASM 2.0 style

```
..> "pong.qasm" "qec.qasm" "cft.qasm" "qft.qasm" >... buffers (φ|tests|φ) python3 build_ast.py ../../examples/qft.qasm -I ../../examples/ -t
28 // quantum Fourier transform
29 OPENQASM 3;
30 include "stdgates.inc";
31 qubit[4] q;
32 bit[4] c;
33 reset q;
34 x q[0];
35 x q[2];
36 barrier q;
37 h q[0];
38 cphase(pi / 2) q[1], q[0];
39 h q[1];
40 cphase(pi / 4) q[2], q[0];
41 cphase(pi / 2) q[2], q[1];
42 h q[2];
43 cphase(pi / 8) q[3], q[0];
44 cphase(pi / 4) q[3], q[1];
45 cphase(pi / 2) q[3], q[2];
46 h q[3];
47 c[0] = measure q[0];
48 c[1] = measure q[1];
49 c[2] = measure q[2];
50 c[3] = measure q[3];
```



# QFT | OpenQASM 3.0 style

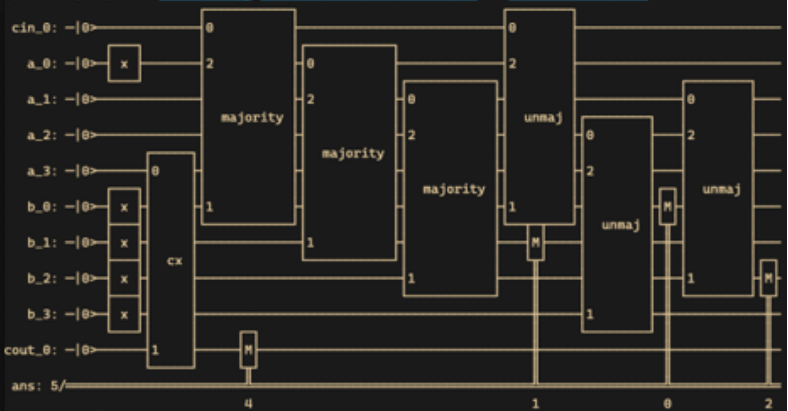
```
..> "qft.qasm" "qft_forloop.qasm" "adder.qasm" > buffers (φ|tests|φ) python3 build_ast.py ../src/openqasm/translator/tests/qft_forloop.qasm -I ../../examples/ -t
21 // quantum Fourier transform
22 OPENQASM 3;
23 include "stdgates.inc";
24 const n = 4;
25 qubit[n] q;
26 bit[n] c;
27 reset q;
28 x q[0];
29 x q[2];
30 barrier q;
31 for i in [0: n-1] {
32   h q[i];
33   for j in [0: i+1] {
34     cphase(pi / (2**(i-j+1))) q[j], q[i+1];
35   }
36 }
37 h q[n-1];
38 c[0] = measure q[0];
39 c[1] = measure q[1];
40 c[2] = measure q[2];
41 c[3] = measure q[3];
```



# Demo Screenshots

# Ripple carry adder | OpenQASM repo test

```
*adder.qasm > alignment.qasm > cphase.qasm > dd.qasm > ... > buffers (|tests|) python3 build_ast.py ../../examples/adder.qasm -i ../../examples/ -t
18 //
17 OPENQASM 3;
16 include "stdgates.inc";
15
14 gate majority a, b, c {
13   cx c, b;
12   cx c, a;
11   ccx a, b, c;
10 }
9
8 gate unmaj a, b, c {
7   ccx a, b, c;
6   cx c, a;
5   cx a, b;
4 }
3
2 qubit[1] cin;
1 qubit[0] a;
22 qubit[0] b;
1 qubit[1] cout;
2 bit[5] ans;
3 uint[4] a_in = 1; // a = 0001
4 uint[4] b_in = 15; // b = 1111
5 // initialize qubits
6 reset cin;
7 reset a;
8 reset b;
9 reset cout;
10
11 // set input states
12 for i in [0: 4] {
13   if(bool(a_in[i])) x a[i];
14   if(bool(b_in[i])) x b[i];
15 }
16 // add a to b, storing result in b
17 majority cin[0], b[0], a[0];
18 for i in [0: 2] { majority a[i], b[i+1], a[i+1]; }
19 cx a[3], cout[0];
20 for i in [2: -1: 0] { unmaj a[i], b[i+1], a[i+1]; }
21 unmaj cin[0], b[0], a[0];
22 measure b[0] -> ans[0];
23 measure b[1] -> ans[1];
24 measure b[2] -> ans[2];
25 measure cout[0] -> ans[4];
```



## Demo Screenshots

QAMP CHECKPOINT #2 |  ISSUE #23

Thank you!