क्वांटम कंट्रोल

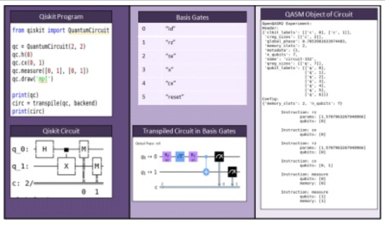
15

क्यूबिट अंशांकन और गेट प्रचालन हेतु क्वांटम सॉफ्टवेयर स्टैक किस्किट का प्रयोक्तानुकूल तरंगरूप जनित्र के साथ समाकलन

*राधिका नासेरी[1], संदीप भराडे[1], एम. वाई. दीक्षित[1], गोपाल जोशी[1] और आर. विजयराघवन[2]

[1]*त्वरक नियंत्रण प्रभाग, भाभा परमाणु अनुसंधान केंद्र (भापअ केंद्र), ट्रांबे-400085, भारत*

[2]*क्वांटम मापन और नियंत्रण (क्यूएमएसी) पसंघनित पदार्थ भौतिकी और सामग्री विज्ञान विभाग, टाटा मूलभूत अनुसंधान संस्थान (टीआईएफआर), मुंबई*



```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0, 1)
qc.measure([0, 1], [0, 1])
qc.draw('mpl')

print(qc)
circ = transpile(qc, backend)
print(circ)
```

0	"id"
1	"rz"
2	"sx"
3	"x"
4	"cx"
5	"reset"

कम्पाइलेशन पाइपलाइन

सारांश

क्वांटम सॉफ्टवेयर स्टैक का एक आर्बिट्रेरी वेवफॉर्म जेनरेटर (एडब्ल्यूजी) के साथ एकीकरण भौतिक क्वैबिट पर क्वांटम एल्गोरिदमऔर क्वांटम प्रयोगोंको निष्पादित करने के लिए महत्वपूर्ण है। यह आलेख ओपन सोर्स किस्किट क्वांटम सॉफ्टवेयर स्टैक को एडब्ल्यूजी के साथ सफलतापूर्वक एकीकृत करने को दर्शाता है, जिसमें किस्किट के विकास के माध्यम से यह हासिल किया गया है।

Quantum Control

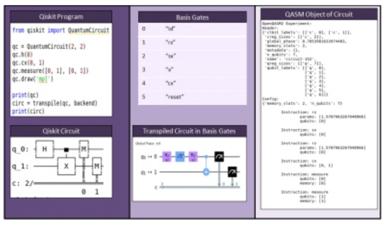
15

Integration of Quantum Software Stack Qiskit to Custom Arbitrary Waveform Generator for Qubit Calibration and Gate Operations

***Radhika Nasery[1], Sandeep Bharade[1], M. Y. Dixit[1], Gopal Joshi[1] and R. Vijayaraghavan[2]**

[1]*Accelerator Control Division, Bhabha Atomic Research Centre (BARC), Trombay-400085, INDIA*

[2]*Quantum Measurement and Control (QuMaC) Laboratory,Department of Condensed Matter Physics and Materials Science, Tata Institute of Fundamental Research (TIFR), Mumbai*



```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0, 1)
qc.measure([0, 1], [0, 1])
qc.draw('mpl')

print(qc)
circ = transpile(qc, backend)
print(circ)
```

Compilation Pipeline

ABSTRACT

Quantum Software Stack integration with a Arbitrary Waveform Generator (AWG) is important for executing quantum algorithms and quantum experiments on physical qubits. This article demonstrates successful integration of Qiskit Quantum Software Stack with AWG through development of Qiskit Backend.

KEYWORDS: *Quantum software stack, Arbitrary waveform generator, Qiskit*

**Author for Correspondence: Radhika Nasery*
E-mail: radhikan@barc.gov.in

Introduction

Quantum computing is rapidly progressing, demanding sophisticated control mechanisms for precise manipulation of quantum bits - qubits. Qubits are the fundamental units of quantum information in quantum computing, and precise control over their states is crucial for implementing quantum gates and executing quantum algorithms. Arbitrary Waveform Generators (AWG) are used to produce user defined, precise and tailored RF waveforms to manipulate the state of qubits. Quantum algorithms are implemented using sequences of quantum gates. AWG support the creation of customizable pulse sequences, enabling the implementation of qubit calibration, quantum experiments and implementation of quantum circuits. To make quantum computers accessible to users and enable the development of quantum applications, a quantum software stack is required. This article presents integration of Qiskit quantum stack with AWG.

Quantum Software Stack

A quantum stack is the hierarchy of components, from the high-level algorithms down to the physical qubits that abstracts away the complexity of the underlying quantum hardware [1]. It typically consists of layers (Fig.1) like a quantum algorithm layer, an intermediate representation layer and a quantum control and readout electronics interface layer. The quantum algorithm layer is where quantum algorithms are implemented using high-level quantum programming languages and software development kits (SDKs). This layer is also used to define qubit calibration experiments. The quantum compilers convert the algorithms written in high level language to a format known as Intermediate Representation (IR). IR layer provides a hardware-agnostic representation of quantum circuits that are compiled and optimized for different target qubit architectures. OpenQASM [2] is an example of a popular IR for quantum circuits. The compiler optimization performs circuit rewriting, transformation, and optimization techniques to map logical qubits in the quantum circuit to the physical qubits of the hardware (as shown in Fig.2), while minimizing circuit depth and gate count. It also synthesizes gates into the native gate set supported by the target hardware. The quantum analog-digital interface layer is responsible for converting the digital gate-level instructions to analog waveforms that control the qubits. This layer handles tasks like frequency control, qubit initialization, gate application, and measurement. A micro-architecture is typically implemented to translate IR for arbitrary waveform generation. This is where arbitrary waveform generators play a key role in precisely shaping the control pulses. The physical qubit control layer

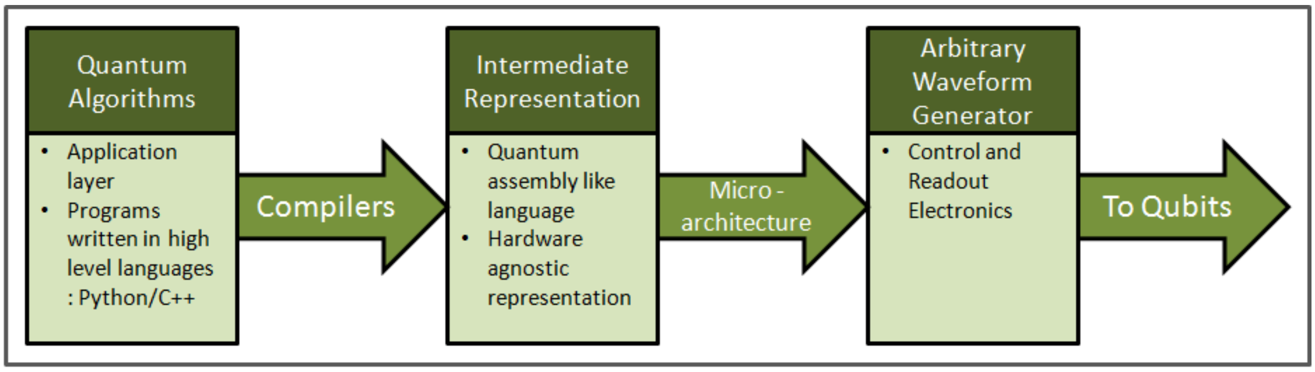
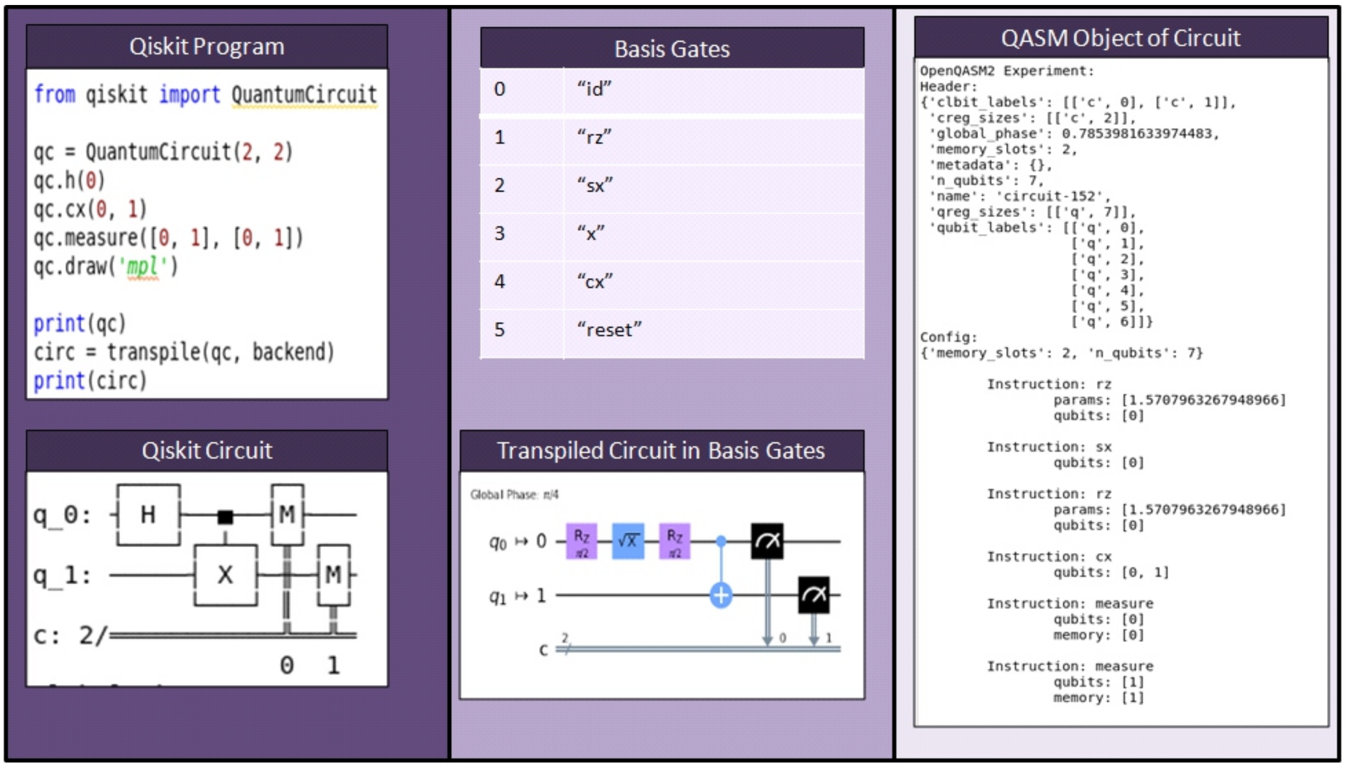

Quantum Algorithms
Application layer
Programs written in high level languages : Python/C++
Compilers
Intermediate Representation
Quantum assembly like language
Hardware agnostic representation
Micro - architecture
Arbitrary Waveform Generator
Control and Readout Electronics
To Qubits

Fig.1: Quantum Software Stack


Qiskit Program

```
from qiskit import QuantumCircuit

qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0, 1)
qc.measure([0, 1], [0, 1])
qc.draw('mpl')

print(qc)
circ = transpile(qc, backend)
print(circ)
```

Basis Gates

0	"id"
1	"rz"
2	"sx"
3	"x"
4	"cx"
5	"reset"

QASM Object of Circuit

```
OpenQASM2 Experiment:
Header:
{'clbit_labels': [['c', 0], ['c', 1]],
 'creg_sizes': [['c', 2]],
 'global_phase': 0.7853981633974483,
 'memory_slots': 2,
 'metadata': {},
 'n_qubits': 7,
 'name': 'circuit-152',
 'qreg_sizes': [['q', 7]],
 'qubit_labels': [['q', 0],
                  ['q', 1],
                  ['q', 2],
                  ['q', 3],
                  ['q', 4],
                  ['q', 5],
                  ['q', 6]]}
Config:
{'memory_slots': 2, 'n_qubits': 7}

        Instruction: rz
                params: [1.5707963267948966]
                qubits: [0]

        Instruction: sx
                qubits: [0]

        Instruction: rz
                params: [1.5707963267948966]
                qubits: [0]

        Instruction: cx
                qubits: [0, 1]

        Instruction: measure
                qubits: [0]
                memory: [0]

        Instruction: measure
                qubits: [1]
                memory: [1]
```

Fig.2: Qiskit Compilation Pipeline.

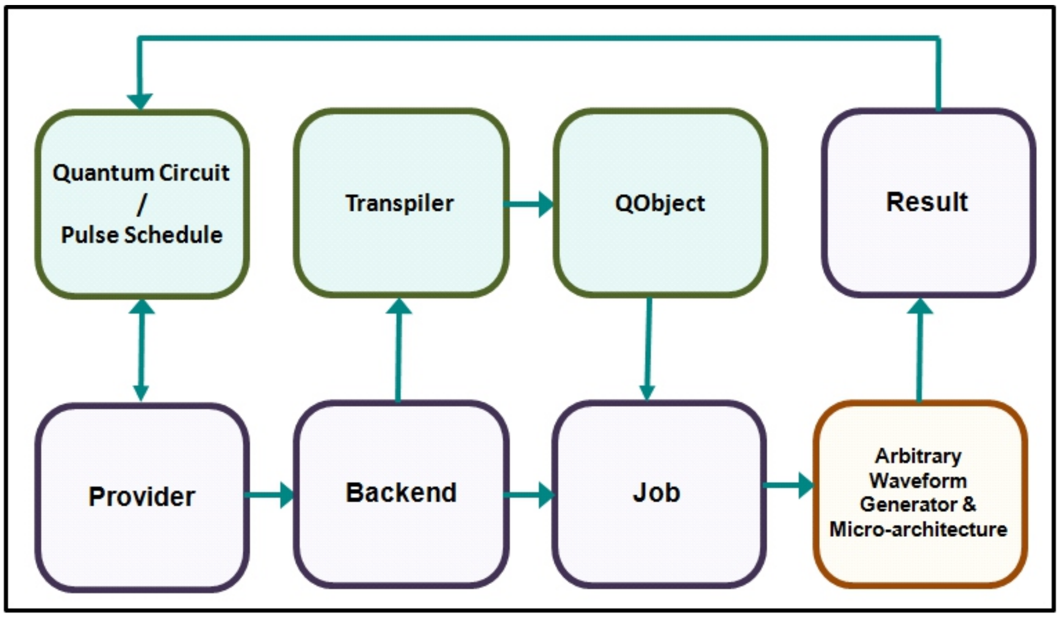

Quantum Circuit / Pulse Schedule
Transpiler
QObject
Result
Provider
Backend
Job
Arbitrary Waveform Generator & Micro-architecture

Fig.3: Qiskit Backend Integration.

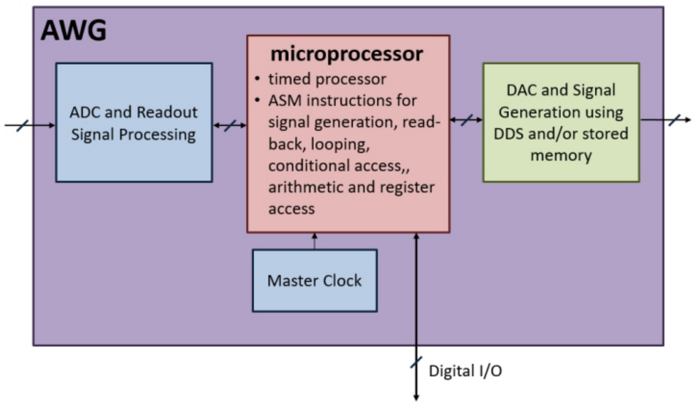

AWG
ADC and Readout Signal Processing
microprocessor
• timed processor
• ASM instructions for signal generation, read-back, looping, conditional access,, arithmetic and register access
DAC and Signal Generation using DDS and/or stored memory
Master Clock
Digital I/O

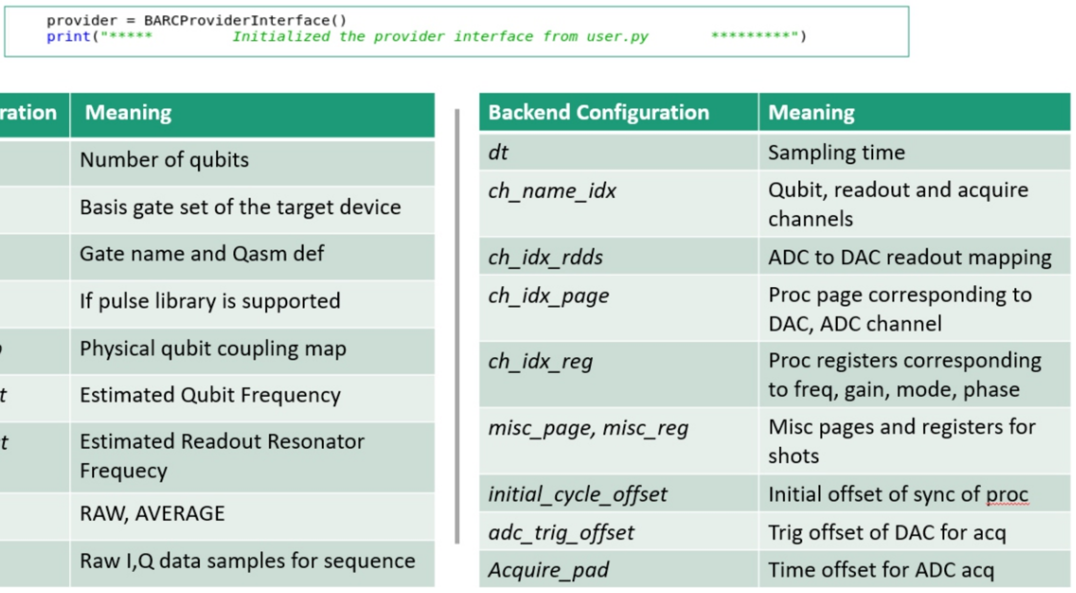
Fig.4: AWG Architecture

interacts directly with the qubits to apply the required operations.

Some of the leading quantum software stacks are Qiskit[3] (IBM), Cirq[4] (Google), Q# [5] (Microsoft), Forest (Rigetti) [6], and Ocean (D-Wave)[7]. While they share many similarities in terms of the overall architecture, they differ in aspects like the supported quantum hardware, programming languages and feature set. Qiskit has emerged as one of the most popular quantum software stacks due to its comprehensive feature set, extensibility, and strong community support. It is an open-source SDK that supports multiple programming languages (Python, Java, Swift, JavaScript), simulators, and quantum hardware backends (IBM, IonQ, Honeywell). Qiskit also provides higher-level abstractions and applications for domains like optimization, machine learning, finance, and chemistry. Therefore, open source Qiskit quantum software stack was selected for integration with AWG. Fig.2 shows the Qiskit compilation pipeline. A quantum circuit is defined in python high level language and submitted for compilation. The compiler transpiles the circuit into basis gates of the quantum hardware and a IR representation in form of OpenQASM2 object is generated.

Development of Qiskit Backend

The Qiskit backend [8] is the interface between the quantum circuits defined in the quantum algorithm layer and the underlying quantum hardware or simulator. It provides a

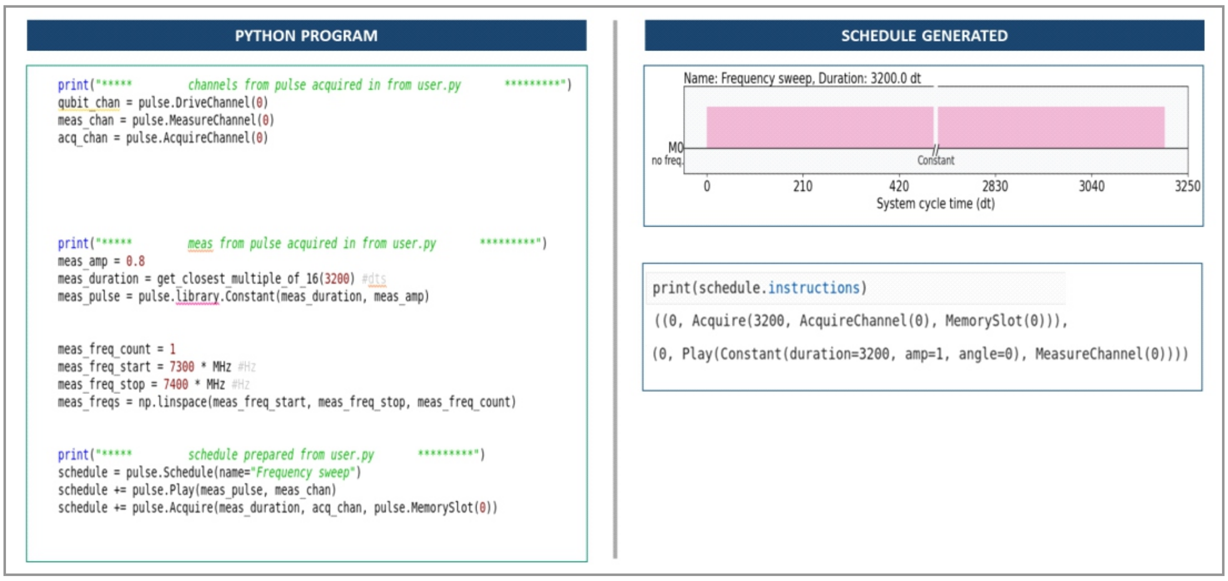


```
provider = BARCProviderInterface()
print("*****        Initialized the provider interface from user.py        **********")
```

Pulse Configuration	Meaning
n_qubits	Number of qubits
basis_gates	Basis gate set of the target device
gates	Gate name and Qasm def
open_pulse	If pulse library is supported
coupling_map	Physical qubit coupling map
qubit_freq_est	Estimated Qubit Frequency
meas_freq_est	Estimated Readout Resonator Frequecy
meas_levels	RAW, AVERAGE
pulse_library	Raw I,Q data samples for sequence

Backend Configuration	Meaning
dt	Sampling time
ch_name_idx	Qubit, readout and acquire channels
ch_idx_rdds	ADC to DAC readout mapping
ch_idx_page	Proc page corresponding to DAC, ADC channel
ch_idx_reg	Proc registers corresponding to freq, gain, mode, phase
misc_page, misc_reg	Misc pages and registers for shots
initial_cycle_offset	Initial offset of sync of proc
adc_trig_offset	Trig offset of DAC for acq
Acquire_pad	Time offset for ADC acq

Fig.5: BARCPoviderInterface and Backend Properties for custom AWG.


PYTHON PROGRAM

```
print("*****        channels from pulse acquired in from user.py       *********")
qubit_chan = pulse.DriveChannel(0)
meas_chan = pulse.MeasureChannel(0)
acq_chan = pulse.AcquireChannel(0)



print("*****        meas from pulse acquired in from user.py       *********")
meas_amp = 0.8
meas_duration = get_closest_multiple_of_16(3200) #dts
meas_pulse = pulse.library.Constant(meas_duration, meas_amp)


meas_freq_count = 1
meas_freq_start = 7300 * MHz #Hz
meas_freq_stop = 7400 * MHz #Hz
meas_freqs = np.linspace(meas_freq_start, meas_freq_stop, meas_freq_count)


print("*****        schedule prepared from user.py       *********")
schedule = pulse.Schedule(name="Frequency sweep")
schedule += pulse.Play(meas_pulse, meas_chan)
schedule += pulse.Acquire(meas_duration, acq_chan, pulse.MemorySlot(0))
```

SCHEDULE GENERATED

```
print(schedule.instructions)

((0, Acquire(3200, AcquireChannel(0), MemorySlot(0))),

(0, Play(Constant(duration=3200, amp=1, angle=0), MeasureChannel(0))))
```

Fig.6: Python Program to provide schedule for single frequency Loopback Testing

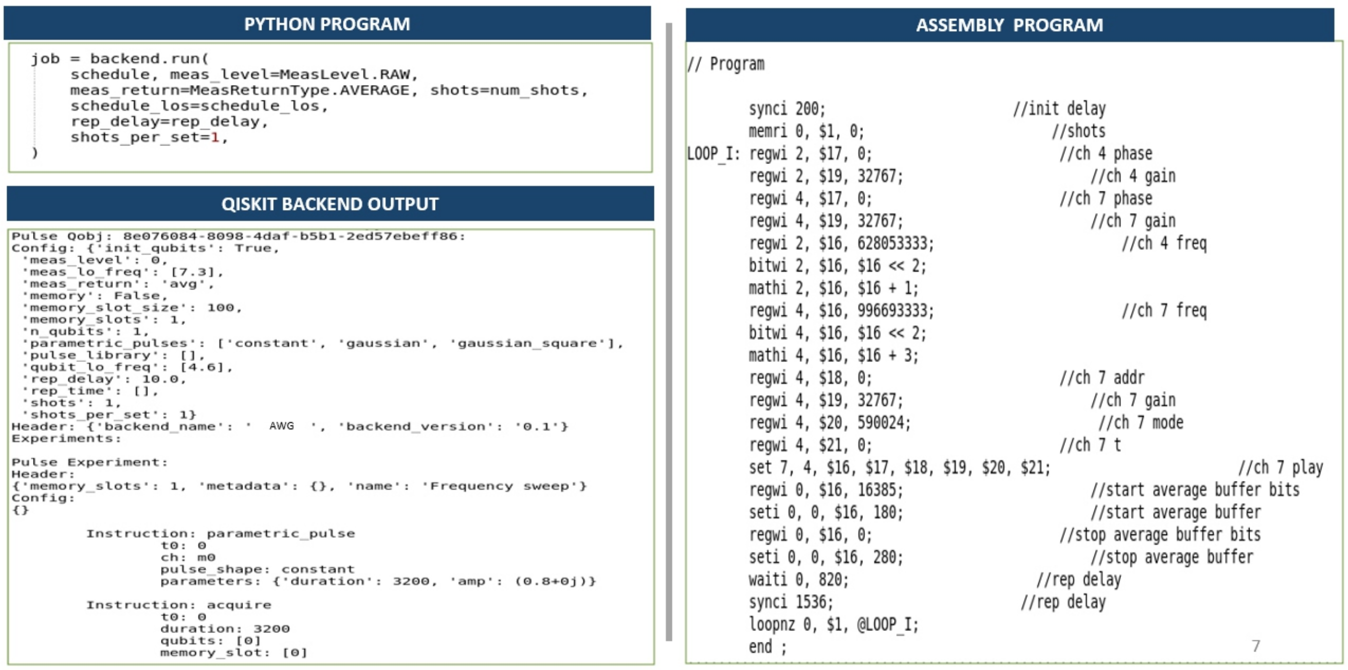
unified API for running quantum circuits on different targets and returning the measurement results. To integrate a AWG into the Qiskit backend, a new (as shown in Fig. 3) backend provider, backend, job and result classes are implemented that conform to the Qiskit backend interface specifications. Quantum circuits are transpiled i.e. optimized by a compiler and then give the output in format of Quantum Object- Qobj. Backends take in a Qobj as input, which is a QASM - IR representation and a Job object is returned. Job instances can be thought of as the "ID" for a submitted job. They find out the execution's state at a given point in time (for example, if the job is queued, running, or has failed) and also allow control over the execution of the job on the AWG. The raw data sample stream from measurement operation of the qubit from the AWG is routed to Result class and then back to the higher level Quantum Circuit/ Pulse Schedule program.

Integration with AWG

The AWG has 3 main parts (Fig.4): micro-processor, readout processing block and signal generating block. The micro-processor with has added timed Assembly Language (ASM) instructions to generate RF pulses. RF pulses are generated for control and readout of qubits. The key aspect of the integration is mapping the quantum gates in the circuit to the corresponding analog waveforms generated by the AWG. This requires a gate-to-pulse mapping given by Fig.5 that translates each quantum gate to a sequence of control pulses with specific amplitudes, frequencies, and durations. The backend provider will use this mapping to convert the quantum circuit into a series of AWG instructions that generate the required pulses. The AWG integration also needs to handle aspects like synchronization, measurement, and feedback between the quantum and classical systems. Fig.5 also shows the python program invoking the implemented "BARCProviderInterface".

Results

Fig.6 shows the Qiskit python program describing the schedule of pulses to be run for a single frequency, constant amplitude loopback testing and corresponding schedule


PYTHON PROGRAM

```
job = backend.run(
    schedule, meas_level=MeasLevel.RAW,
    meas_return=MeasReturnType.AVERAGE, shots=num_shots,
    schedule_los=schedule_los,
    rep_delay=rep_delay,
    shots_per_set=1,
)
```

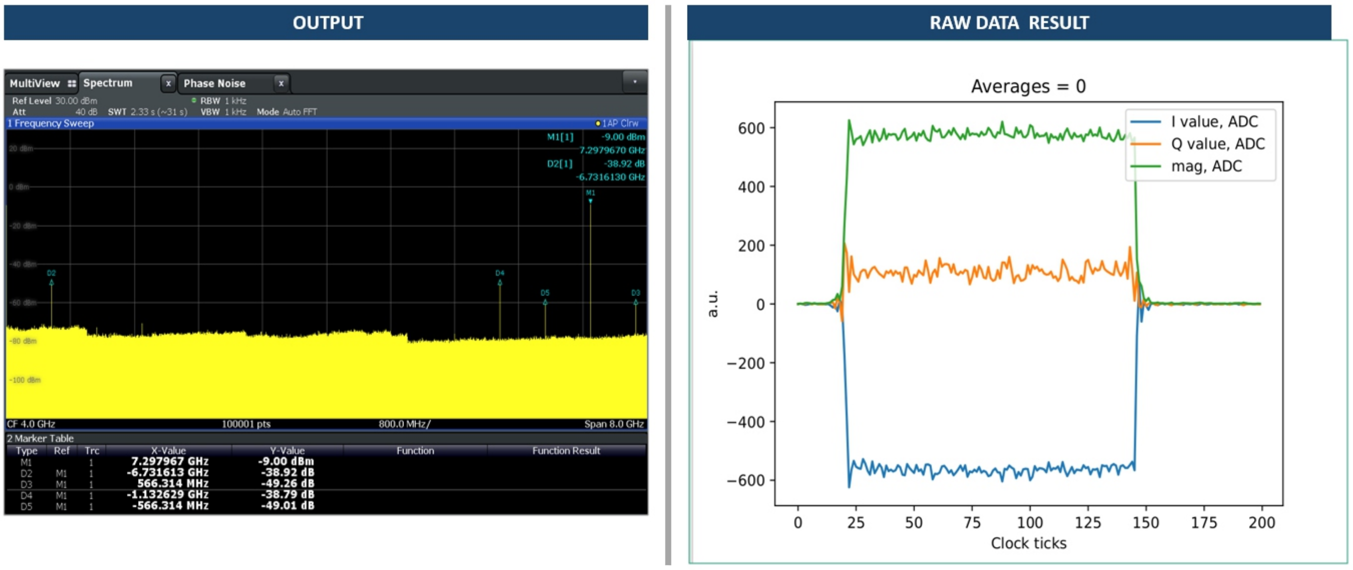
QISKIT BACKEND OUTPUT

```
Pulse Qobj: 8e076084-8098-4daf-b5b1-2ed57ebeff86:
Config: {'init_qubits': True,
 'meas_level': 0,
 'meas_lo_freq': [7.3],
 'meas_return': 'avg',
 'memory': False,
 'memory_slot_size': 100,
 'memory_slots': 1,
 'n_qubits': 1,
 'parametric_pulses': ['constant', 'gaussian', 'gaussian_square'],
 'pulse_library': [],
 'qubit_lo_freq': [4.6],
 'rep_delay': 10.0,
 'rep_time': [],
 'shots': 1,
 'shots_per_set': 1}
Header: {'backend_name': '  AWG  ', 'backend_version': '0.1'}
Experiments:

Pulse Experiment:
Header:
{'memory_slots': 1, 'metadata': {}, 'name': 'Frequency sweep'}
Config:
{}

        Instruction: parametric_pulse
                t0: 0
                ch: m0
                pulse_shape: constant
                parameters: {'duration': 3200, 'amp': (0.8+0j)}

        Instruction: acquire
                t0: 0
                duration: 3200
                qubits: [0]
                memory_slot: [0]
```

ASSEMBLY PROGRAM

```
// Program

        synci 200;                      //init delay
        memri 0, $1, 0;                      //shots
LOOP_I: regwi 2, $17, 0;                      //ch 4 phase
        regwi 2, $19, 32767;                      //ch 4 gain
        regwi 4, $17, 0;                      //ch 7 phase
        regwi 4, $19, 32767;                      //ch 7 gain
        regwi 2, $16, 628053333;                      //ch 4 freq
        bitwi 2, $16, $16 << 2;
        mathi 2, $16, $16 + 1;
        regwi 4, $16, 996693333;                      //ch 7 freq
        bitwi 4, $16, $16 << 2;
        mathi 4, $16, $16 + 3;
        regwi 4, $18, 0;                      //ch 7 addr
        regwi 4, $19, 32767;                      //ch 7 gain
        regwi 4, $20, 590024;                      //ch 7 mode
        regwi 4, $21, 0;                      //ch 7 t
        set 7, 4, $16, $17, $18, $19, $20, $21;                      //ch 7 play
        regwi 0, $16, 16385;                      //start average buffer bits
        seti 0, 0, $16, 180;                      //start average buffer
        regwi 0, $16, 0;                      //stop average buffer bits
        seti 0, 0, $16, 280;                      //stop average buffer
        waiti 0, 820;                      //rep delay
        synci 1536;                      //rep delay
        loopnz 0, $1, @LOOP_I;
        end ;                                               7
```

Fig.7: Python Program to run a pulse schedule and ASM output for AWG.



2 Marker Table

Type	Ref	Trc	X-Value	Y-Value	Function	Function Result
M1		1	7.297967 GHz	-9.00 dBm		
D2	M1	1	-6.731613 GHz	-38.92 dB		
D3	M1	1	566.314 MHz	-49.26 dB		
D4	M1	1	-1.132629 GHz	-38.79 dB		
D5	M1	1	-566.314 MHz	-49.01 dB		

Fig.8: RF Frequency Spectrum Output and Results of Loopback Testing in Qiskit Python Program.

generated. Qubit channel is the channel for qubit state manipulation. Measure Channel is the channel for readout resonator pulse generation and Acquire Channel is the waveform acquisition channel for Qubit state measurement.

Fig.7 shows this Schedule running as a job on the Backend. The Backend compiles this Schedule as a Qiskit Backend Object and generates ASM language program to be run on the AWG using language to language parser. Generated ASM program is executed on micro-processor. The DAC and ADC are loop-backed.

Fig.8 shows the generated frequency of 7.3GHz as seen on Spectrum Analyzer. The tproc reads the ADC measurement channel and this data is displayed as Qiskit Result.

Conclusion

Qiskit Backend is successfully developed for AWG. The Backend integrates the Qiskit stack with the analog RF pulse generation. Python program generating a Qiskit pulse schedule of a single frequency loopback testing is successfully demonstrated on the backend. Future work involves integration of Qiskit experiment library with Backend. The AWG Backend for Qiskit software stack can be installed at intranet facility so that users across organization can avail this facility for quantum experiment.

Acknowledgments

We sincerely appreciate QuMaC Lab, TIFR, for their critical role in advancing this research. Their exceptional support and expertise have significantly contributed to the outcomes of this study.

References

[1] Frederic T. Chong, Diana Franklin & Margaret Martonosi, “Programming languages and compiler design for realistic quantum hardware”, Nature 549, 180–187 (2017). https://doi.org/10.1038/nature23459

[2] Andrew W. Cross, Lev S. Bishop, John A. Smolin, Jay M. Gambetta, “Open Quantum Assembly Language”, arXiv:1707.03429v2[quant-ph], 13 Jul 2017

[3] Gadi Aleksandrowicz, et al. “Qiskit: An Open-source Framework for Quantum Computing. 0.7.2” , Zenodo, 23 Jan. 2019, doi:10.5281/zenodo.2562111.

[4] Cirq Developers, “Cirq”. Zenodo, Dec. 01, 2023. doi: 10.5281/zenodo.10247207.

[5] A. S. Tolba, M. Z. Rashad, and Mohammed A. El-Dosuky. 2013 “Q#, a quantum computation package for the .NET platform”. arXiv preprint arXiv:1302.5133

[6] Peter J. Karalekas, et al. “Pyquil: Quantum Programming in Python. v2.17.0 ”, Zenodo, 30 Jan. 2020, doi:10.5281/zenodo.3631770.

[7] “D-Wave's Ocean Software,” http://ocean.dwavesys.com (2020)

[8] IBM Quantum Documentation, API Reference, Providers Interface, https://docs.quantum.ibm.com/api/qiskit/qiskit.providers.Backend.