Creating custom gates using qiskit pulse

Soumik Samanta[*1], Ayan Barui[2]

[1]*Department of Physics, St. Paul's Cathedral Mission College, University of Calcutta, Kolkata, India*

[2]*Department of Physics, Central University of Haryana, Haryana, India*

September 13, 2022

Abstract

A Quantum Circuit is a model for quantum computation, in which a computation is a sequence of quantum gates, measurements, initialization of qubits to known values, and possibly other actions. Quantum Circuit model allows user to only work at circuit level thus hiding the underlying physical implementation of quantum gates and measurements. Qiskit Pulse is a low level pulse-programming paradigm implemented as a module within Qiskit Terra. This gives user precise control over working at real quantum hardware level, the ability to calibrate and execute pulse, and read-out level instructions. In this paper we use Qiskit Pulse model to calibrate and create gates with high fidelity. We start by calibrating the qubit frequency of qubit 0 at IBMQ Armonk backend, accurately by frequency sweep method, which will later be implemented in preparing pulse schedule for our custom gate in section 5. We perform calibration of single qubit Hadamard Gate by defining and varying the pulse parameters, in different cloud-based IBM backends to higher fidelity. Finally, we create a single qubit custom gate 'C' and 2-qubit custom bellgate 'BG', and display the results from IBMQ Armonk and IBMQ Belem respectively.

Keywords: schedule, transpile, backend, instruction set, calibration, fidelity

[*]soumiksamanta10@gmail.com

Contents

1 Introduction

Quantum mechanics is a mathematical framework or set of rules for the construction of physical theories which provides more general and fundamental description of nature than classical mechanics. Further, Quantum computing is an area of computing focused on developing computation of complex systems based on the principles of quantum theory such as superposition and entanglement. As classical computers are built from electrical circuit containing wires and logic gates which works on bits, the quantum computers on the other hand are built from *quantum circuits* and elementary *quantum gates* which works on *quantum bits* (qubits). Qubits are the basic unit of computation in quantum computing which can exist in a superposition of 0 and 1 providing greater computational advantage over the classical bits. Quantum computers have proved their supremacy over classical supercomputers over the recent years.

Qiskit is an open source software development kit developed by IBM for working with quantum computers at the level of circuits, algorithms, and application modules. Qiskit-Pulse is a low-level programming language providing user with higher precision control over quantum hardware than circuit level programming. It is a module used for specifying pulse level control over a real quantum device independent of the specific hardware implementation[2].

In the experiments shown in this article cloud based IBM Quantum Systems and Fake-Backends were used.

1.1 Superconducting qubits

To interact with quantum computers a physical type of qubit called superconducting qubit is required such as electron spin in silicon, ultracold atoms, ion traps etc. To create solid-state qubits, we need to isolate a two level quantum system. A key component in most of these superconducting qubits is a device called josephson junction in which the thin layer of aluminium oxide acting as an insulator is sandwiched between two superconducting layer of aluminium, which becomes superconductor when cooled below 1.2 K [7]. Two energies are important while designing a superconducting qubit. The Josephson energy, E_j, is a measures the strength of coupling across the junction and the Coulomb charging energy, E_c, is required to increase the charge on a junction by 2e. The junction is nothing but a capacitor with $E_c = 4e^2/(2c)$, where e is the charge of electron and C is the capacitance. Three types of qubits can be created using Josephson junction viz. charge qubits, flux qubits and phase qubits[6].

1.2 Qubit representation and control

As qubits exists in a superposition of 0 and 1, they can be represented as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where α and β being complex numbers with $|\alpha|^2+|\beta|^2 \leq 1$. Qubits can also be represented in another form as,

$$|\psi\rangle = \cos(\theta/2)\,|0\rangle + \exp(\iota\phi)\sin(\theta/2)\,|1\rangle$$

where θ and ϕ define a point on the unit three-dimensional sphere known as *Bloch sphere*. With the Bloch sphere it is possible to visualize the state of a single qubit as shown in figure 1[9].

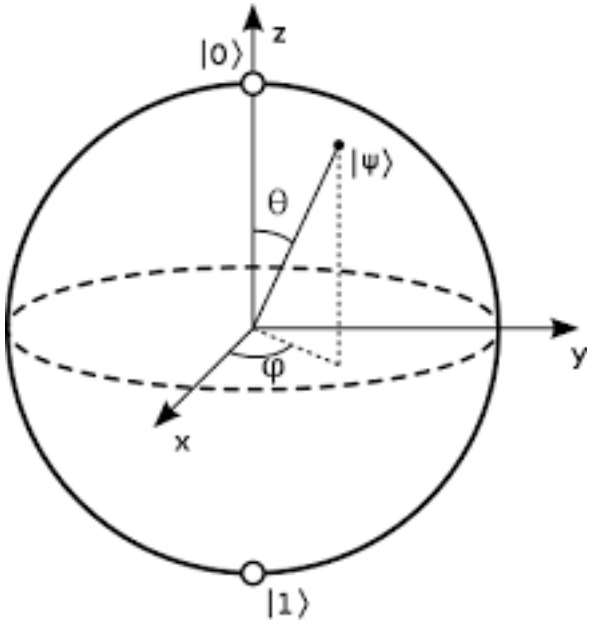


Figure 1: Bloch sphere representation of a qubit

To control the qubits, the control electronics is used which sends electromagnetic signals i.e. pulses to change the state of qubit by performing unitary operations.

1.3 Pulse programming

1.3.1 Pulse

Electromagnetic signals of varying amplitude (Complex valued, $[u_0, u_1 \ldots u_{n-1}]$ and unit norm are called pulses. Each u_j can be named as samples. Sample rate is a cycle-time of dt which is the finest time-resolution exposed on the pulse coprocessor for a specific system. The mathematical form of of the ideal output is,

$$D_j = Re[\exp(i2\pi f jdt + \phi)d_j]$$

where f is frequency, phi is phase of the modulation and jdt represents time. We can define parametric pulse shapes. As an example we can look at the Gaussian Pulse from

the Pulse Library. It uses three parameters: time duration dt, complex amplitude amp, and standard deviation sigma.

```
from qiskit.pulse.pulse_lib import Gaussian

duration = 128
amp = 0.2
sigma = 16
gaussian_pulse = Gaussian(duration, amp, sigma)
```

Figure 2: Pulse Import Code

A pulse specifies an arbitrary pulse envelope. The first part of the figure below describes the envelopes as an input given to the Arbitrary Waveform Generator (AWG - A lab instrument). The signal produce by AWG is then combined with continuous sine wave generator in which frequency of the output can be set. At last, signal sent to the qubit is depicted in the last part of the figure below.

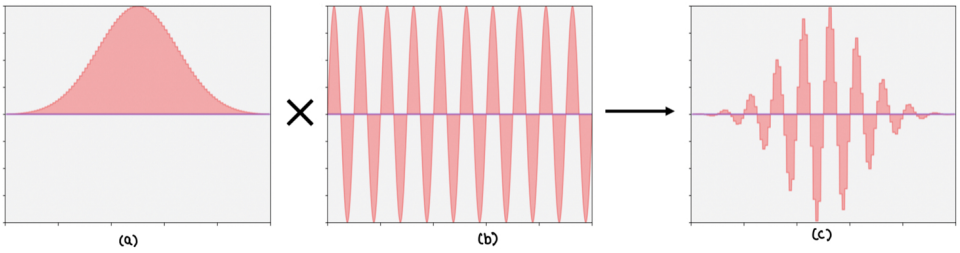


Figure 3: Intuition of frequency modulation

1.3.2 Channels

Channels connect us with qubits. Through channels we send pulses to qubits. Channels are basically abstract models of hardware components., they label the signal lines to transmit or receive signals between the control electronic hardware and the quantum device. Each channel has a specific associated Hamiltonian term H_k with respect to the hardware of the system [10].

Channel	Alias	Description
PulseChannel	-	sub-types are *DriveChannel, MeasureChannel and ControlChannel. Generic transmit channel* used to manipulate the quantum system.
DriveChannel	d_i	Transmit channel connected to qubit i , with signals typically modulated at a frequency in resonance with qubit i.
MeasureChannel	m_i	Transmit channel connected to the readout component of qubit i.
ControlChannel	u_i	Transmit channel with signals typically associated with arbitrary interaction terms in the Hamiltonian.
AcquireChannel	a_i	Receive channel connected to the readout component of qubit i, capable of digitizing and acquiring data.

Through table 1 one can understand the basic description or uses of the Pulse channel model[5].

1.3.3 Instructions and Schedules

Instructions are used to manipulate Quantum Systems via channels. An allocation and trigger timing model is followed by IBM.

Instruction	Operand	Description
Play	pulse: *Pulse*, channel: *PulseChannel*	Output the waveform described by pulse on the channel.
Delay	duration: *int*, channel: *Channel*	Idle the channel for the given duration.
ShiftPhase	phase: *float*, channel: *PulseChannel*	Shift the phase of the channel by phase radians.
Acquire	duration: *int*, channel: *AcquireChannel*, register: *Register*	Trigger the channel to collect data for the given duration, and store the measurement result in a register

Delay Instruction is supported in every channel, which has a operands such as duration which is specified as a number of cycles, and a target channel.. The play instruction enables users to play a pulse on a target Pulse Channel with a specific frequency and phase set determined by Shift Phase and Set Frequency instructions. In the Acquire instruction we have a duration, and Acquire Channel, and a classical register to store the observed result. This instruction assigns the measurement unit to begin Acquire data and for how long.

The Pulse schedule is a collection of ordered instructions. Pulse schedules can be fixed

with with the order and duration of the instructions.

```
# Create a pulse schedule.
sched = Schedule(name='excited_state')

# Create gate and measurement pulses.
x180 = Drag(x_dur, x_amp, x_sigma, x_beta)
measure = GaussianSquare(m_dur, m_amp, m_sigma, m_square_width)

# += appends an Instruction to a Schedule.
sched += Play(x180, DriveChannel(0))

# Measure qubit 0.
sched += Play(measure, MeasureChannel(0))

# Determine the state of qubit 0 and store it
# in a persistent MemorySlot register which

# will be returned in the program result.
sched += Acquire(AcquireChannel(0), MemorySlot(0))

# Run the schedule and get the result.
counts = execute(sched, backend).result().get_counts()
```

Figure 4: Schedule Code

1.3.4 Scheduler

Quantum circuits and Pulse Schedulers are both different repetitions of Quantum Program. A scheduler translates a Circuit program to Pulse Program.

Every IBM device has a particular topology and native universal gate set. Based on these properties, Qiskit Transpiler optimises the quantum circuits to run on a specific device.

It's important to note that Pulse programs operate on physical qubits. A drive pulse on qubit 0 will not enact the same logical operation on the state of qubit 1 – in other words, gate calibrations are not interchangeable across qubits. This is in contrast to the circuit level, where an X gate is defined independent of its qubit operand.

```
qc = QuantumCircuit(2, 2)
qc.h(1)
qc.cx(1, 0)
qc.measure([0, 1], [0, 1])
qc = transpile(qc, backend)
pulse_schedule = schedule(qc, backend)

# Plot the program representations.
qc.draw()
pulse_schedule.draw()
```

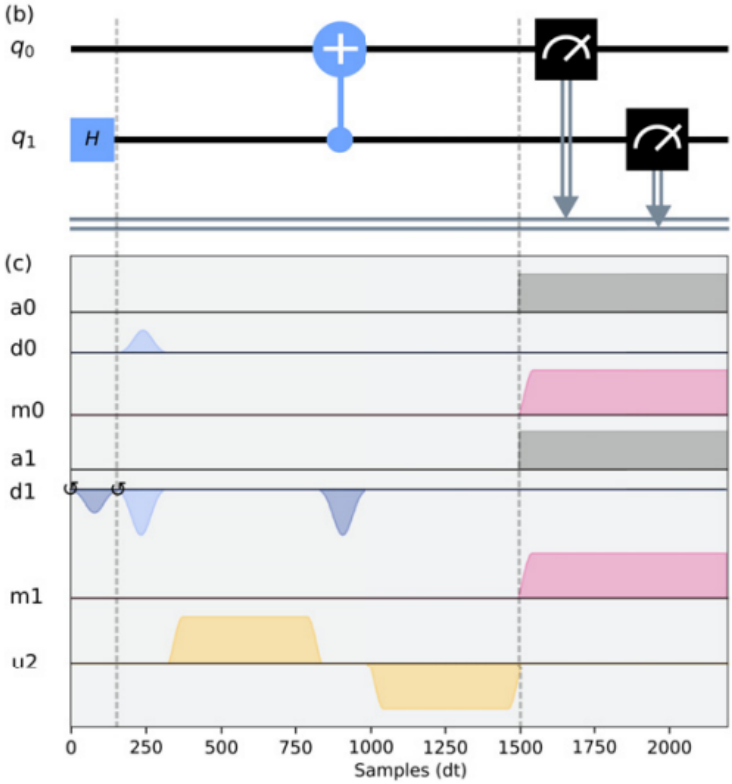
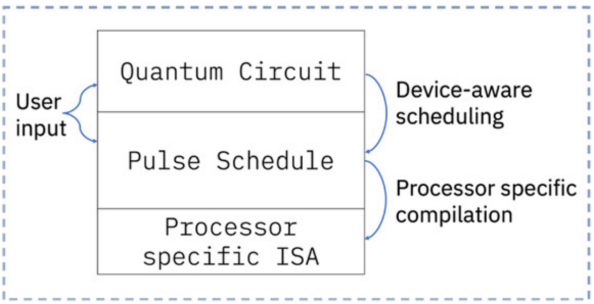


Figure 5: Transition of Circuit to Pulse Code



2 Calibrating Qubits with Pulse gates

Pulse gates provide the user with more control over the low-level hardware control of the qubits. Pulse gates allow us to map a logical circuit gate to Qiskit pulse program called *schedule*[1]. This mapping is called as *Calibration*. Calibration is an optimized pulse level description of operation on qubit(s). Schedules describe instruction sequences

for control hardware. It specifies the exact time dynamics of the input signals across all input channels to the device. This interface is more powerful and requires a deeper understanding of the underlying device physics.
Let us try to find the exact calibrated frequency of the qubit 0 in IBMQ Armonk backend. Qubit frequency is the energy difference between the ground state $|0\rangle$ and the excited state $|1\rangle$. This qubit frequency will be later useful for modulating the waveform in building pulse schedules.
In a typical lab setting, the qubit frequency can be found by sweeping a range of frequencies. To define the frequency range that will be swept for a qubit, we have restricted the sweep range to a window of 100Hz in steps of 2Hz around the default frequency of the backend. The default frequency of qubit 0 of Armonk was found to be 4.971677465384175 GHz. We then start the sweep, looking for a peak in the absorption curve with the help of a tool called Network Analyzer. We define a *sweepgate* with frequency as the parameter varying in the range 4.9216774653841755GHz to 5.021677465384175GHz in steps of 0.002GHz.

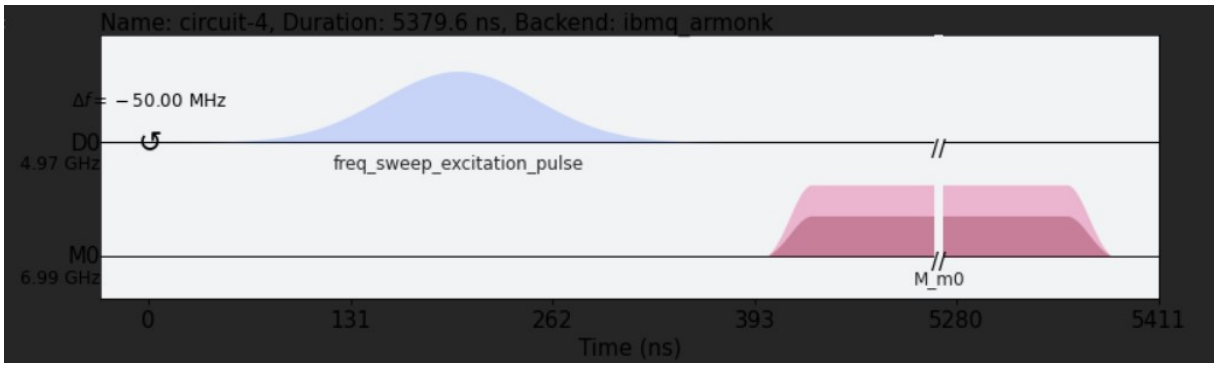


Figure 6: Frequency sweep schedule

In the figure 6, the blue curve represents the drive pulse (frequency sweep excitation pulse) acting on drive channel D0. The red one represents the measurement pulse acting on measurement channel M0.

Plotting the measured signal for driving frequency and doing a Lorentzian curve fit for the plot results in figure 7. Measuring the corresponding frequency value for the peak, the qubit's frequency can be updated from 4.97168 GHz to 4.97164 GHz.

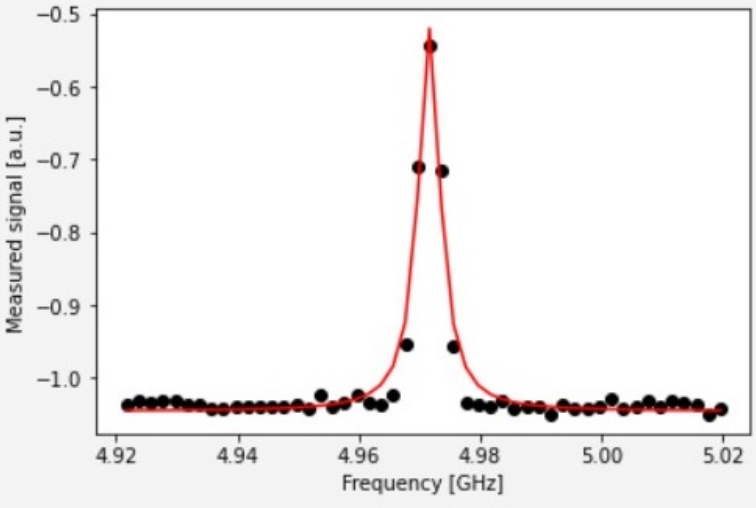


Figure 7: Frequency fit curve

3 Calibrating Hadamard Gate on Simulator FakeArmonk

Backends that are OpenPulse enabled will typically have calibrations defined for measurements and for each of its basis gates. Calibrations can also be defined or updated by the user through pulse schedules. Let us calibrate a single qubit gate, the *Hadamard* gate in FakeArmonk backend. We proceed by adding a hadamard gate to a quantum circuit and measuring it. Transpiling the circuit and using the scheduler to convert the transpiled quantum circuit into pulse schedule, default schedule instructions are obtained. Now, we try to calibrate the above hadamard gate, by custom defining its pulse schedule. This can be done with the help of pulse builder. It contextually constructs a pulse schedule and then emits the schedule for execution. User can optionally pass backend into build() to enable enhanced functionality.

Here are the default parameters of Hadamard gate for FakeArmonk:
duration=320 dt
amp=0.37526098415448106-0.08060535276562332j
σ*=80*
beta=-0.9829945644928844

Applying the pulse with above parameters and phase shifts as below:
Shift Phase(-Pi/2,channel) ..... #Shifting phase
pulse.Play(h, channel) ..... #applying the pulse
Shift Phase(-Pi/2,channel) ..... #shifting to default phase

The default Hadamard gate schedule is shown in figure 8.

Now, let the custom parameters of the Hadamard gate be:
duration=256dt
amp=0.2
σ*=50*

On calibrating this pulse schedule to the hadamard gate, the schedule of the circuit is obtained as in figure 9.

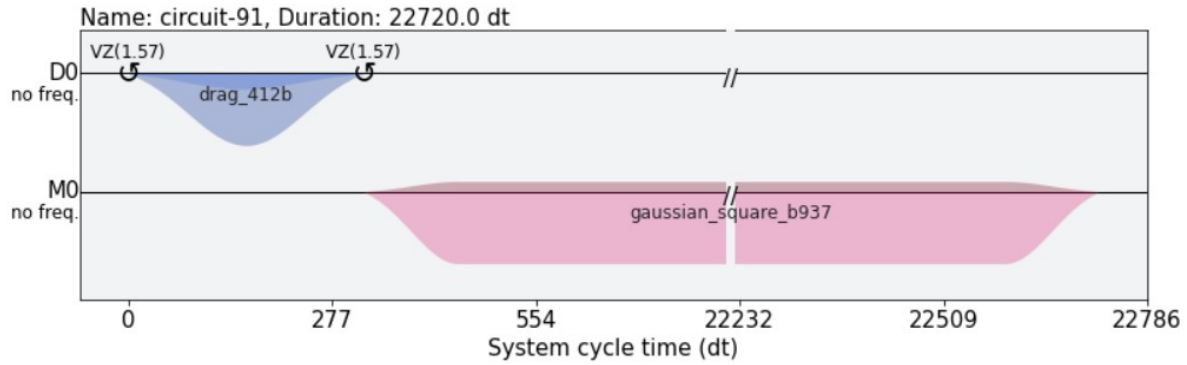


Figure 8: Default Hadamard schedule

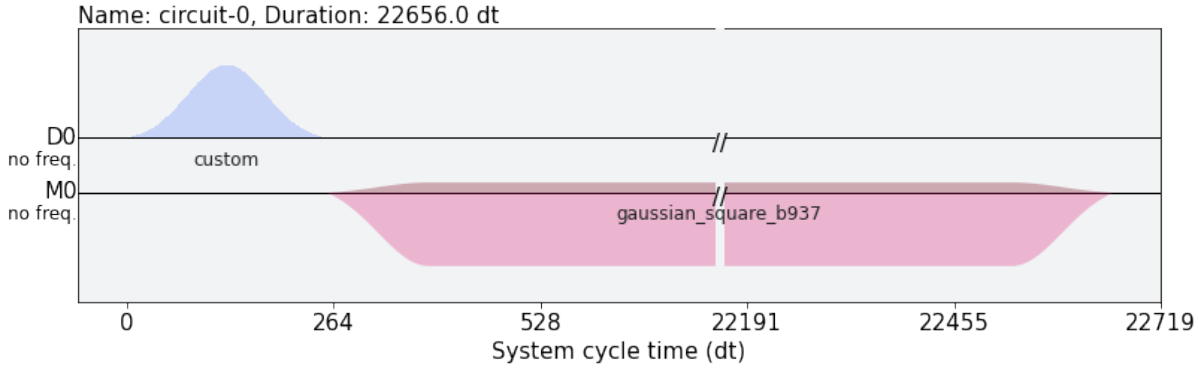


Figure 9: Calibrated Hadamard schedule

Notice that the initial pulse on D0, the Hadamard gate, is now implemented with our custom pulse.

4 Calibrating Hadamard Gate on IBMQ Nairobi and IBMQ Quito

Real backends have sources of error or noise which affect the desired outcomes of the experiment. The gates one can apply in a quantum computer are parameterized by variables, for ex. in case of superconducting qubits, applying a gate implies driving the qubit with a microwave pulse from arbitrary waveforms. The amplitude, sigma, duration, beta of this pulse are the parameters and these are all subject to some error. This implies neither the gate nor the resultant state are not going to be the way we wanted. Access to pulse level programming renders the user to calibrate the pulse by varying these parameters so as to improve the fidelity of the gate.
Calibrating the gates is also one of the method to modify the existing gates, to get the desired results according to the user's needs.
To create a custom gate for real backend with high fidelity, we employ the same method as in section 3. Defining the default parameters of a gate directly at the hardware end rather than applying the same gate at circuit level also results in increase in its fidelity. First we employ the method to find the schedule for Hadamard gate, for IBMQ Nairobi backend.

Here are the default parameters for IBM Nairobi qubit 0:
duration=160
amp=0.06989063434283332+2.8058840562449595e-05j
sigma=40
beta= -0.39651044278480513
name="H"

Applying the pulse with above parameters and phase shifts as below:
Shift Phase(-Pi/2, channel) ..... #Shifting phase
pulse.Play(h, channel) ..... #applying the pulse
ShiftPhase(-Pi/2, channel) ..... #shifting to default phase

Feeding the schedule into the circuit we get a calibrated Hadamard gate for IBMQ Nairobi. The calibrated pulse is shown in figure 10.

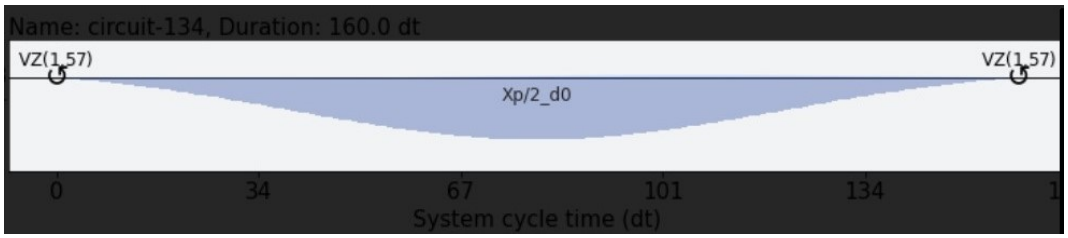


Figure 10: Calibrated Hadamard Pulse for IBM Nairobi Qubit 0

The results for 20000 shots are as follows:

Counts for $|0\rangle$ are 10218
and for $|1\rangle$ are 9782

State Matrix is represented as $\begin{pmatrix} 0.5109 & 0.4434 - 0.02055i \\ 0.4434 + 0.02055i & 0.4891 \end{pmatrix}$

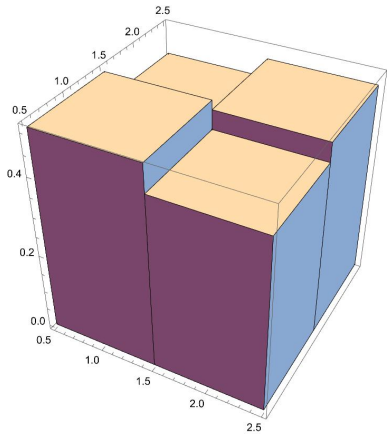


Figure 11: State City Plot for IBM Nairobi Qubit (0) calibrated Hadamard

Fidelity of this gate would be of 0.997601. This is comparable to the default fidelity of 0.997274 the Hadamard Gate on the same Qubit, hence, we would consider this experiment successful.

Repeating the experiment for IBMQ Quito 3^{rd} qubit, we get following results for 20000 shots:
Counts for $|0\rangle$ are 11149
and for $|1\rangle$ are 9399

State Matrix is represented as $\begin{pmatrix} 0.55745 & 0.37005 - 0.03005i \\ 0.37005 + 0.03005i & 0.44255 \end{pmatrix}$

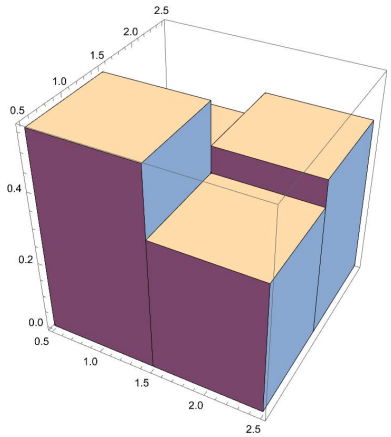


Figure 12: State City Plot for IBM Quito Qubit (3) calibrated Hadamard

Fidelity would be of 0.9837. This fidelity is as expected is comparable to the default fidelity of 0.977328 of 3^{rd} Qubit of IBMQ Quito.
Thus we have successfully created (high fidelity) Hadamard Gate using Pulses on 1^{st} qubit of IBM Nairobi and 3^{rd} qubit of IBMQ Quito.

5 Custom gate on IBMQ Armonk

Creating a non-standard, custom gate also employs the same procedure as discussed before. Let us try to build a custom gate 'C' using the IBMQ backend Armonk[3][8]. First let us define the custom gate pulse schedule of parametric Gaussian waveform with following parameters:
duration = 320,
amplitude = 0.112233 + 0.778899j,
$\sigma = 80$
*pulse.set frequency (4.97164 *GHz,channel)* #modulating the waveform with the calibrated qubit frequency from section 2.
The resulting waveform is represented in figure 13.
Calibrating the above pulse schedule to the 'C' gate appended to qubit 0 and measuring it, gives figure 14.

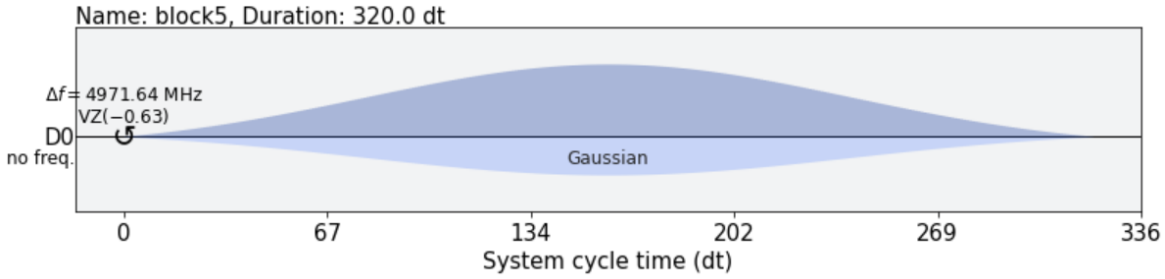


Figure 13: Custom Gaussian Waveform

Figure 14

The undefined custom gate is transpiled and with scheduler the pulse schedule of the circuit can be obtained as shown in figure 15.

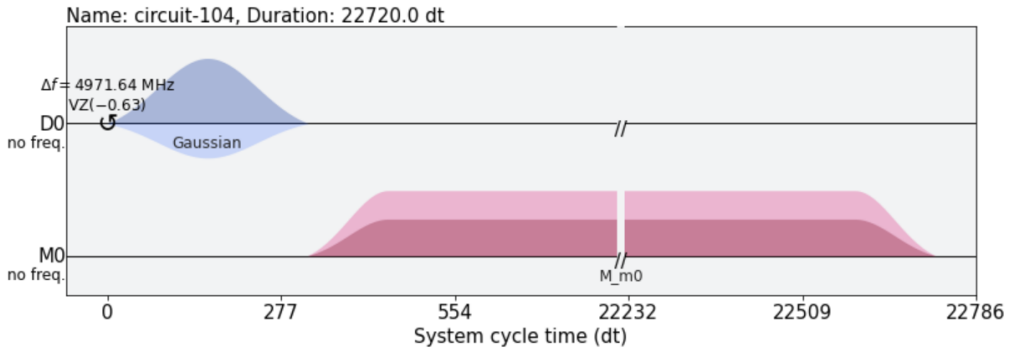


Figure 15: Custom gate schedule

We then assemble the pulse schedule to the backend and run the job with 3200 shots. The result of the job gave the following data.:
Number of counts: $['0' : 336, '1' : 2864]$
i.e. the resultant state was $|0\rangle$ 336 times and the state was $|1\rangle$ 2860 times.
Also, visualization of these counts can be seen as a histogram plot in figure 16.

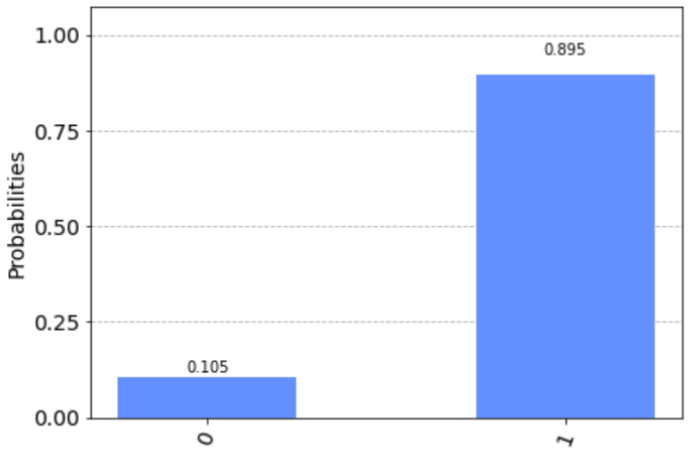


Figure 16: Plot of results of C gate

In case of using a FakeBackend, the PulseSimulator and PulseSystemModel of the corresponding backend, has to be called to support the pulse schedule as input for the job execution.

6 Two qubit Custom gate on IBMQ Belem

We now try to create a two qubit custom gate using qiskit pulse. For this experiment we use IBMQ Belem having 5-qubit configuration, as our backend. We have optimised the circuit for the first two qubits of the system.

Our goal for this experiment is to combine 'H' and 'CNot' gate into a single custom gate so as to get bell state $|\beta_{00}\rangle$. For this we use the schedule instructions from the circuit with both gates[4]. Combining the pulses gives us schedule as in figure 17.

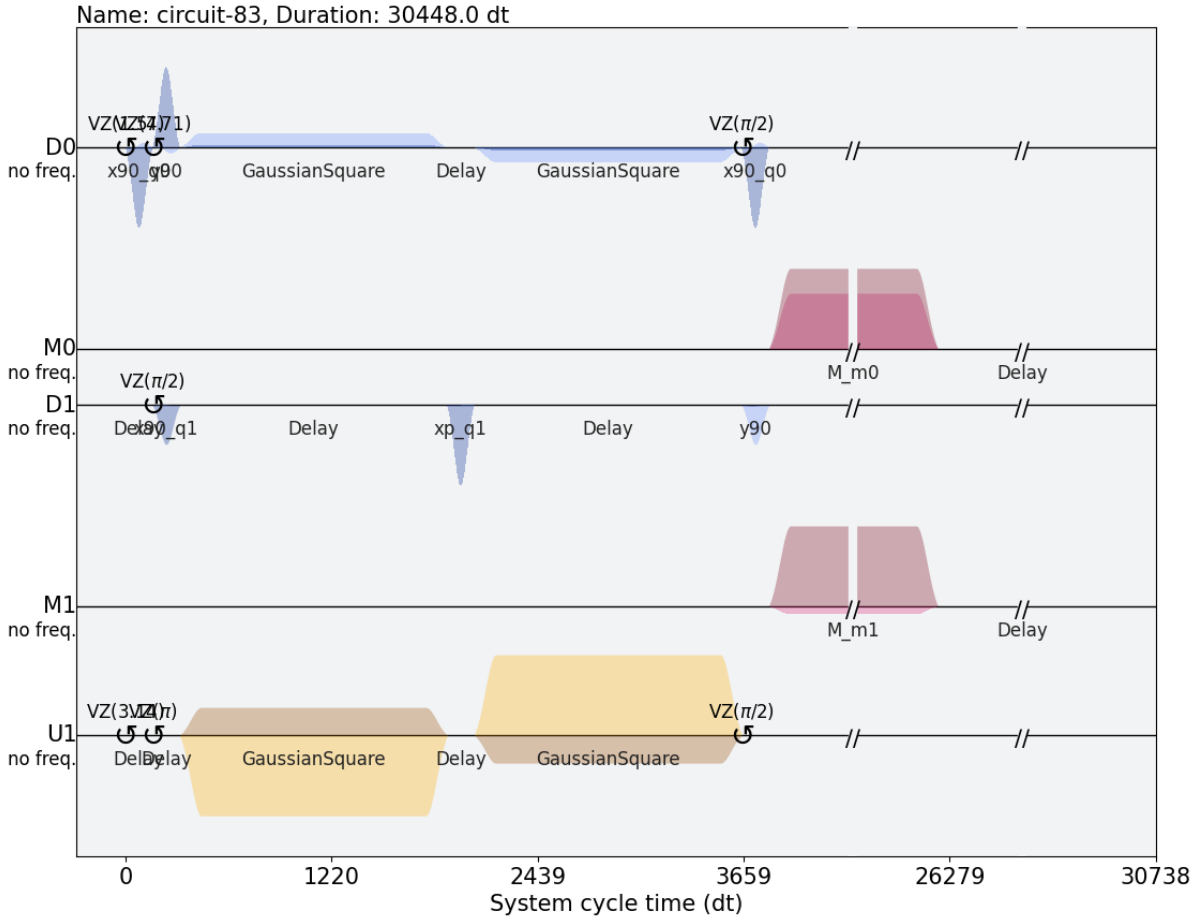


Figure 17: Pulse schedule for custom Bell Gate.

On mapping the schedule instructions to a custom 2- qubit gate in a quantum circuit, would result in figure 18.

Assembling the circuit and running the job on IBMQ Belem for 2000 shots. We obtained the following result. Number of counts recorded were:

$|00\rangle$: 970
$|01\rangle$: 69
$|10\rangle$: 65
$|11\rangle$: 896

Visualization of counts as histogram is shown in figure 19.

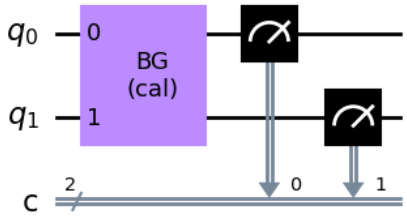


Figure 18: Circuit of custom Bell Gate.

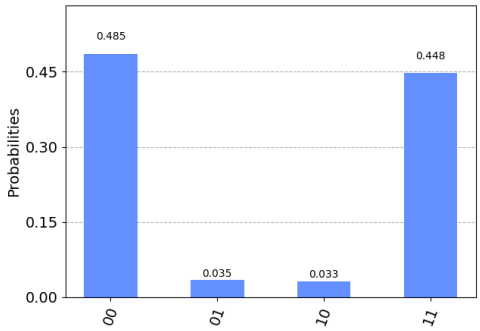


Figure 19: Probability plot of output state of BG(custom gate).

As expected we have created bell state

$$\frac{1}{\sqrt{2}}(|00> + |11>)$$

with fairly high accuracy using pulse level programming. Thus this custom gate is as accurate as default combination of Hadamard and CNot gate and our experiment to combine two gates and create a custom 2-qubit gate was a success.

7 Finding gate parameters for desired statevector

If the user desires an resultant state $|\psi\rangle$, from the known initial state of the qubit, one can construct a quantum gate which does the exact above transformation. The following algorithm is to be implemented for the same.
Step 1: Initialize the desired resultant state on the qubit.
Step 2: transpile and schedule the circuit,and
Step 3: Use *schedulename.instructions* to get the parameters of this transformation.

We have demonstrated an example: Our desired output state is $(\sqrt{2/3}\,|0\rangle + \sqrt{1/3}\,|1\rangle)$. Following the algorithm, the instruction set of the transformation is as shown in figure 21. This instruction set can further be used to build pulse schedule and then to calibrate it to a prepared custom gate as discussed in section 4 and 5.

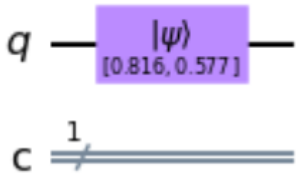


Figure 20: Quantum circuit with initialized gate

```
((0, ShiftPhase(1.5707963268, DriveChannel(0))),
 (0,
  Play(Drag(duration=320, amp=(0.37526098415448106-0.08060535276562332j), sigma=80, beta=-0.9829945644928844,
name='drag_412b'), DriveChannel(0), name='drag_412b')),
 (320, ShiftPhase(-1.5707963268, DriveChannel(0))))
```

Figure 21: Instruction set

References

[1] Building pulse schedules — qiskit 0.37.0 documentation. https://qiskit.org/documentation/tutorials/circuits_advanced/06_building_pulse_schedules.html. (Accessed on 07/16/2022).

[2] Calibrating qubits with qiskit pulse. https://qiskit.org/textbook/ch-quantum-hardware/calibrating-qubits-pulse.html. (Accessed on 07/16/2022).

[3] Using qiskit pulse to create a custom gate, display that gate's microwave, then execute the circuit with a pulse simulator. · github. https://gist.github.com/splch/42143a394a7e7a71a05186e12eb7b2d1. (Accessed on 07/16/2022).

[4] Using the scheduler — qiskit 0.37.0 documentation. https://qiskit.org/documentation/tutorials/circuits_advanced/07_pulse_scheduler.html. (Accessed on 07/16/2022).

[5] Thomas Alexander, Naoki Kanazawa, Daniel J Egger, Lauren Capelluto, Christopher J Wood, Ali Javadi-Abhari, and David C McKay. Qiskit pulse: Programming quantum computers through the cloud with pulses. *Quantum Science and Technology*, 5(4):044006, 2020.

[6] Michel H Devoret, Andreas Wallraff, and John M Martinis. Superconducting qubits: A short review. *arXiv preprint cond-mat/0411174*, 2004.

[7] Morten Kjaergaard, Mollie E. Schwartz, Jochen Braumüller, Philip Krantz, Joel I.-J. Wang, Simon Gustavsson, and William D. Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11(1):369–395, mar 2020.

[8] David C. McKay, Thomas Alexander, Luciano Bello, Michael J. Biercuk, Lev Bishop, Jiayin Chen, Jerry M. Chow, Antonio D. Córcoles, Daniel Egger, Stefan Filipp, Juan Gomez, Michael Hush, Ali Javadi-Abhari, Diego Moreda, Paul Nation, Brent Paulovicks, Erick Winston, Christopher J. Wood, James Wootton, and Jay M. Gambetta. Qiskit backend specifications for openqasm and openpulse experiments, 2018.

[9] Michael A Nielsen and Isaac L Chuang. Quantum computation and quantum information. *Phys. Today*, 54(2):60, 2001.

[10] Nandhini R. Improving quantum gates using pulse programming. http://dr.iiserpune.ac.in:8080/xmlui/handle/123456789/6950, May 2022. (Accessed on 07/16/2022).