

Qiskitチュートリアル勉強会 7

Qiskit Pulse

梅沢 和正



目的

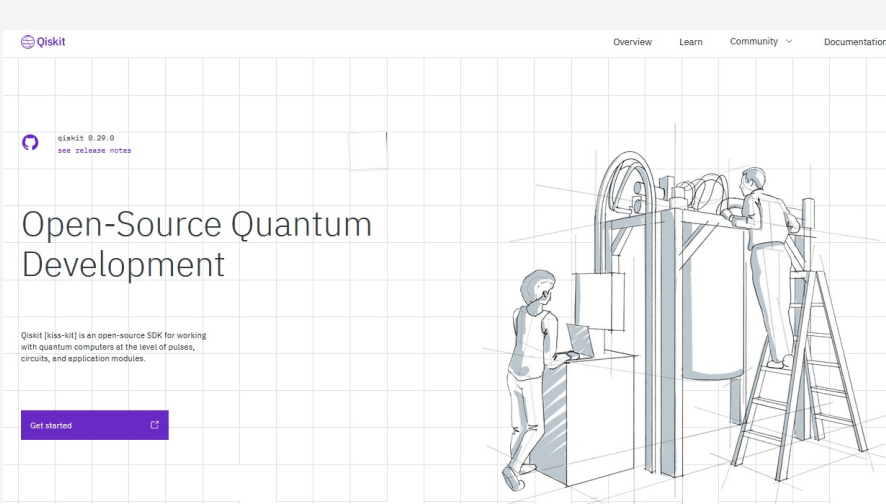
Builder構文を理解する。

アジェンダ

1. はじめに
2. Builder構文
3. 以前のコードとの比較
4. Scheduleの便利な機能
5. 量子回路とSchedule
6. Parameterクラスを使ったBuilder構文

はじめに ~Qiskit~

Qiskitは、量子コンピュータ用のオープンソースのフレームワーク

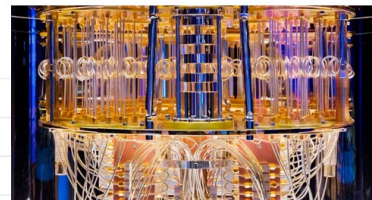
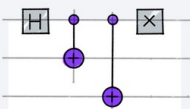


What can Qiskit do

Qiskit accelerates the development of quantum applications by providing the complete set of tools needed for interacting with quantum systems and simulators.

Access to circuits

Access a rich set of well-studied circuits, which can be used as benchmarks, building blocks in more complex circuits, or as a tool to explore quantum computational advantage.

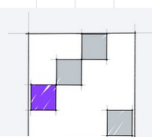


Hardware Access

Execute code on multiple quantum hardware architectures, from superconducting qubits to trapped-ions.

Noise Mitigation

Study and reduce the impact of noise using built-in modules for noise characterization and circuit optimization.



Quantum Algorithms

Research and prototype machine learning, optimization, and chemistry applications by building upon a library of quantum algorithms.



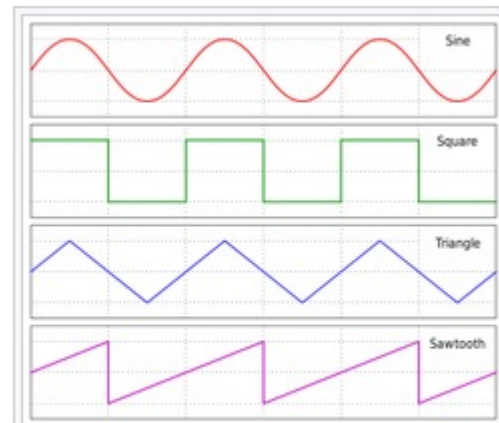
<https://qiskit.org/>

API	説明
Qiskit (Terra)	量子機械語水準か、それに近い量子回路を作成するツールを提供する
Qiskit Simulator (Aer)	ノイズの影響を再現などシミュレーターを提供する
Qiskit Experiments (Ignis)	デバイスのベンチマーク、エラーの軽減、エラー修正のためのツールが含まれる
Qiskit Application Modules (Aqua)	ユーザー自身が量子プログラミングを行うことなく使用できるツールを提供する。現在、化学、AI、最適化、金融の分野がサポートされる https://qiskit.org/documentation/locale/ja_JP/aqua_tutorials/Qiskit%20Algorithms%20Migration%20Guide.html
Qiskit IBM Quantum (Provider)	アカウントやバックエンドデバイスなどを管理するクラスを提供する。

はじめに ~パルスとは~

パルスは、短時間に急峻な変化をする信号の総称

- ✓ 電子回路の分野では一定の幅を持った矩形波のことをパルスといい、クロック信号や同期信号に使われる。
- ✓ 黒電話など回線ダイヤル式の電話機から電話交換機へ送出する選択信号をダイヤルパルスという。
- ✓ パルスの変化によって信号を変調することをパルス変調という。パルス幅変調、パルス振幅変調、パルス符号変調などの種類がある。
- ✓ シンセサイザーの音色では、矩形波以外に三角波やのこぎり波もパルス波という。

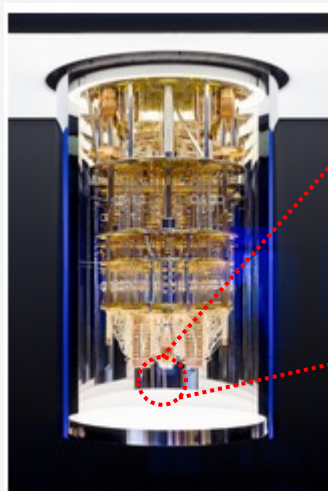


(上から) 正弦波、矩形波、三角波、のこぎり波の波形

<https://ja.wikipedia.org/wiki/%E3%83%91%E3%83%AB%E3%82%B9>

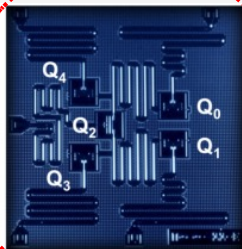
はじめに ～超電導の量子コンピュータ～

超電導の量子コンピュータはパルスを当てて制御している

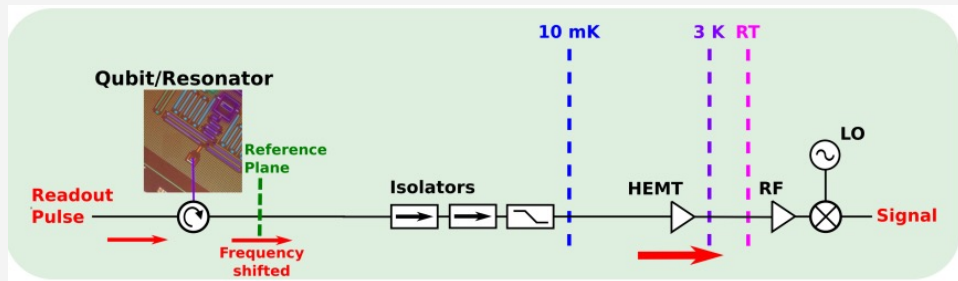


希釈冷凍機のインサート部分

https://qiskit.org/documentation/locale/ja_JP/qc_intro.html



量子ビットを接続する2つのバス共振器と制御にも使用される5つの読み出し共振器で構成される5つの量子ビットプロセッサの顕微鏡写真。*)



読み出しパルス図

読み出しパルスは、希釈冷凍機の入力ラインを介して超伝導キュービット/共振器システムに適用されている。**)

*),**)ともに

Fast, high-fidelity readout of multiple qubits

N T Bronn, B Abdo, K Inoue, S Lekuch, A D C 'orcoles, J B Hertzberg, M Takita, L S Bishop, J M Gambetta, J M Chow

IOP Conf. Series: Journal of Physics: Conf. Series 834 (2017) 012003

はじめに ~With構文~

何かの処理の開始時と終了時に必須の処理をしてくれる構文のこと

ファイル処理コード

```
▶ with open("hoge.txt","r") as fileread:  
    print(fileread.read())
```

with構文で利用できるクラスには、`__enter__`、`__exit__`といったメソッドがある。

この二つのメソッドがwith構文で利用できるために必要なメソッドとなる。

- ✓ `__enter__`メソッドはwithブロック内で呼び出され、戻り値には自身のインスタンスを返却する必要がある。
- ✓ `__exit__`メソッドはwithブロックを抜けたときに呼び出される。また、例外が発生した場合にもこのメソッドは呼ばれるため、例外処理もここに記述する。

はじめに ~量子回路の作り方~

量子回路とは、*量子データ(量子ビットなど)に対するコヒーレントな量子演算と、リアルタイム同時古典計算で構成される計算ルーチン*のこと

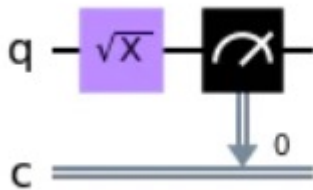
(参考サイト)

<https://qiskit.org/textbook/ja/ch-algorithms/defining-quantum-circuits.html>

```
sample_sx = QuantumCircuit(1, 1)
sample_sx.sx(0)
sample_sx.measure(0, 0)
```

```
sample_sx.draw(output='mpl')
```

QuantumCircuitクラスを作成
ゲートを登録
測定を登録
全体の量子回路を表示



Builder構文

パルス・プログラムは、Scheduleと呼ばれ、制御エレクトロニクスの命令シーケンスを記述する。Pulse BuilderはそのScheduleを作成する。

```
from qiskit import pulse

GHz = 1.0e9 # Gigahertz
mem_slot=0
freq_GHz=4.95
freq=freq_GHz*GHz

# 1 Builder構文の定義
with pulse.build(backend) as sample_sched:
    # 2 フローの定義
    with pulse.align_sequential():
        # 3 周波数の定義
        pulse.set_frequency(freq, pulse.drive_channel(qubit))
        pulse.play(pulse.Gaussian(duration=320,
                                   amp=1.0,
                                   sigma=80,
                                   name='sample gaussian'),
                   pulse.drive_channel(qubit))

# 4 測定パルスの定義
pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])
```

#	説明
1	パルスプログラミングを開始するには、最初にビルダー構文をbuild()で初期化する必要がある。 (参考サイト https://qiskit.org/documentation/stubs/qiskit.pulse.builder.build.html#qiskit.pulse.builder.build)
2	align_sequential()を利用してパルスのスケジュールをシーケンシャルに実行するように宣言している。他にもalign_left()、align_right()が提供されており自由に組み合わせることで多彩なパルスのフローを設定することが可能。 with pulse.build(backend=backend, default_alignment='sequential', name='sample') as sample_sched: 上記の設定と同じ。
3	set_frequency()を利用して周波数を定義できる。
4	パルスレベルでは、測定はパルスと、システム測定ユニットにデータを取得して処理するように指示する取得命令の両方で構成されている。measure()は、プロセスを自動化するために提供されているが、必要に応じて、acquire()およびplay()を使用して完全に制御することも可能。

以前のコードとの比較

Builder構文を利用して、プログラム表現（データ）をプログラミング構文（コード）から分離できるようになった。

従来の書き方

```
from qiskit import pulse
from qiskit.pulse import DriveChannel
from qiskit.pulse import Play
from qiskit.pulse import Schedule

qubit=0

# 1 パルスの定義
drive_pulse = pulse.Gaussian(duration=300,
                              sigma=80,
                              amp=1.0,
                              name='sample gaussian')

# 2 チャンルの指定
drive_chan = pulse.DriveChannel(qubit)

# 3 測定パルスの定義
inst_sched_map = backend_defaults.instruction_schedule_map
measure = inst_sched_map.get('measure', qubits=backend_config.meas_map[0])

# 4 スケジュールを作成
schedule = pulse.Schedule(name='Sample')
schedule += Play(drive_pulse, drive_chan)
schedule += measure << schedule.duration
```



Builder構文

```
from qiskit import pulse

GHz = 1.0e9 # Gigahertz
mem_slot=0
freq_GHz=4.95
freq=freq_GHz*GHz

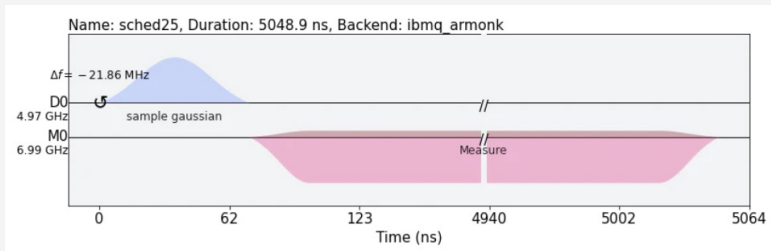
# 1 Builder構文の定義
with pulse.build(backend) as sample_sched:
    # 2 フローの定義
    with pulse.align_sequential():
        # 3 周波数の定義
        pulse.set_frequency(freq, pulse.drive_channel(qubit))
        pulse.play(pulse.Gaussian(duration=320,
                                    amp=1.0,
                                    sigma=80,
                                    name='sample gaussian'),
                    pulse.drive_channel(qubit))
    # 4 測定パルスの定義
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])
```

Scheduleの便利な機能

定義したスケジュールを確認する方法として、`draw()`を利用する、`instructions`を利用するなどがある。

① パルススケジュールの可視化

```
sample_sched.draw(backend=backend)
```



横軸は時間、縦軸はパルスを表していてD0は駆動パルス、M0は測定パルスを表現してる。(0Iは一つ目)

このフローは、周波数が設定されその後、Gaussianパルスとmeasureパルスが実行されるスケジュールになっていることが分かる。

ここで`draw()`に`backend`を渡しているがこれはどの`backend`を利用するか分からないので必要で、なにも指定しない場合`dt`という内部データを利用した値でパルスが表示される。

② パルス設定の表示

```
sample_sched.instructions
```

```
((0, SetFrequency(4950000000.0, DriveChannel(0))),  
(320,  
 Play(Gaussian(duration=320, amp=(1+0j), sigma=80, name='sample gaussian'), DriveChannel(0), name='sample gaussian')),  
(640, Acquire(22400, AcquireChannel(0), MemorySlot(0))),  
(640,  
 Play(GaussianSquare(duration=22400, amp=(-0.3584733362723958+0.05040701520361846j), sigma=64, width=22144, name='gaussian_square_b937'), MeasureChannel(0), name='gaussian_square_b937')))
```

`instructions`を使えば自分が定義したフローを①とは違った見方で確認できる。
自分が指定した内容が一覧で閲覧できるので便利

デモ1



量子回路とSchedule

ほとんどの量子アルゴリズムは回路演算だけで記載できるが、プログラムの低レベル実装をより制御する必要がある場合は、パルスゲートを使用する。

```
from qiskit import pulse

GHz = 1.0e9 # Gigahertz
qubit=0
freq_GHz=4.95
freq=freq_GHz*GHz
drive_sigma_sec = 0.075e-6
drive_duration_sec = drive_sigma_sec * 8
drive_amp = 0.3

# 1. カスタムゲート作成
x_pulse = Gate('x_pulse', num_qubits=1, params=[])

# 2. 1で作成したカスタムゲートを含む量子回路を作成
sample = QuantumCircuit(1, 1)
sample.append(x_pulse, [0])
sample.measure(0, 0)

#3. 自分が作成したいパルスの定義を記述したscheduleクラスを作成
with pulse.build(backend) as x_schedule:
    pulse.set_frequency(freq, pulse.drive_channel(qubit))
    pulse.play(pulse.Gaussian(duration=16 * int(pulse.seconds_to_samples(drive_duration_sec) /
                                         amp=drive_amp,

#4. 3で作成したクラスを2で定義したゲートにバインド
sample.add_calibration(x_pulse, (0, ), schedule=x_schedule)
```

#	説明
1	自分で定義したユニタリゲートを作成することができる。 (参考サイト https://qiskit.org/documentation/stubs/qiskit.circuit.Gate.html)
2	1で作成したカスタムゲートを含む量子回路を構築する。
3	Builder構文を利用してパルス进行を定義する。ここではパルスモジュールで提供されているメソッドseconds_to_samples()を利用している。このメソッドは引数に秒を与えることで、アクティブなバックエンドで秒単位で経過するサンプルの数を取得することができます。 (参考サイト https://qiskit.org/documentation/stubs/qiskit.pulse.builder.seconds_to_samples.html)
4	自分が作成したパルスとゲートをバインドする。

デモ2



Parameterクラスを使ったBuilder構文

変数を制御フロー（Schedule）と別に定義できることから、Scheduleを使いまわすことが可能で再利用性が高くなっている

```
import numpy as np

GHz = 1.0e9 # Gigahertz
MHz = 1.0e6 # Megahertz

# サンプル数は16の倍数である必要があります
def get_closest_multiple_of_16(num):
    return int((num + 8) // 16 * 16)

qubit=0
mem_slot=0

# 1 周波数領域の決定
center_frequency_Hz = backend_defaults.qubit_freq_est[qubit]
frequency_span_Hz = 40 * MHz
frequency_step_Hz = 1 * MHz
frequency_min = center_frequency_Hz - frequency_span_Hz / 2
frequency_max = center_frequency_Hz + frequency_span_Hz / 2
frequencies_GHz = np.arange(frequency_min / GHz,
                             frequency_max / GHz,
                             frequency_step_Hz / GHz)

# 2 パルスに必要な値の設定
drive_sigma_sec = 0.075e-6
drive_duration_sec = drive_sigma_sec * 8
drive_amp = 0.05

# 3 Parameterクラスの定義
freq = Parameter('freq')
# 4 Builder構文
with pulse.build(backend=backend, default_alignment='sequential') as parametrized_sched:
    # 5 Parameterクラスで定義した変数を周波数に設定
    pulse.set_frequency(freq, pulse.drive_channel(qubit))
    pulse.play(pulse.Gaussian(duration=get_closest_multiple_of_16(pulse.seconds_to_samples(drive_duration_sec)),
                              amp=drive_amp,
                              sigma=pulse.seconds_to_samples(drive_sigma_sec),
                              name='spect_pulse'), pulse.drive_channel(qubit))
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])

# 6 周波数領域とParameterクラスを使って定義したScheduleをバインド
frequencies_Hz = frequencies_GHz*GHz
schedules = [parametrized_sched.assign_parameters({freq: f}, inplace=False) for f in frequencies_Hz]
schedules[0].draw(backend=backend)
```

#	説明
1	sweepする周波数を設定する。ここでは中心となる周波数の値をbackend_defaultsから取得している。
2	パルスに必要な値の設定を行う。
3	Parameterクラスを定義する。ここでは変数となる値freqをクラスに渡している。
4	Builder構文を利用してScheduleを作成する。
5	set_frequency()を利用してScheduleで利用する周波数を定義する。ここでは実際の値を定義するのではなくParameterクラスを定義しておく。
6	周波数領域とParameterクラスを使って定義したScheduleをバインドする。

低エネルギーと高エネルギーで周波数をスweepしたい

変数「周波数」「振幅」をParameterクラスに登録して使う。

```
freq = Parameter('freq')
amp = Parameter('amp')
with pulse.build(backend=backend, default_alignment='sequential') as parametrized_sched2:
    pulse.set_frequency(freq, pulse.drive_channel(qubit))
    pulse.play(pulse.Gaussian(duration=get_closest_multiple_of_16(pulse.seconds_to_samples(drive_duration_sec)),
                              amp=amp,
                              sigma=pulse.seconds_to_samples(drive_sigma_sec),
                              name='spect_pulse'), pulse.drive_channel(qubit))
    pulse.measure(qubits=[qubit], registers=[pulse.MemorySlot(mem_slot)])
```

① 低エネルギー

```
# 周波数領域と振幅(low power)をParameterクラスを使って定義したScheduleとバインド
frequencies_Hz = frequencies_GHz*GHz
schedules = [parametrized_sched2.assign_parameters({freq: f, amp: 0.1}, inplace=False) for f in frequencies_Hz]
```

② 高エネルギー

```
# 周波数領域と振幅(high power)をParameterクラスを使って定義したScheduleとバインド
frequencies_Hz = frequencies_GHz*GHz
schedules = [parametrized_sched2.assign_parameters({freq: f, amp: 1.0}, inplace=False) for f in frequencies_Hz]
```

同じフローのScheduleを複数書かなくても良く便利である。

デモ3

まとめ

- ✓ Qiskit PulseではBuilder構文を使ってScheduleを作成する。
- ✓ Qiskit Pulseを使って低レベル実装をより制御することができる。
- ✓ Parameterクラスを併用することでScheduleの再利用性が高めることができる。

梅沢 和正