Single-Q-calibration

July 14, 2025

1

`qlisp` `Scanner2`

`Notebook`

2

2.1

$|\psi(t)\rangle$ $U(t)$

$$i\frac{\partial}{\partial t}|\psi(t)\rangle = H|\psi(t)\rangle|\psi'\rangle = U|\psi\rangle U(t) = \mathcal{T}\exp\left[-i\int_0^t H(t')\mathrm{d}t'\right]$$

two-level system (TLS) ,

$$H \equiv \begin{pmatrix} a+z & x-iy \\ x+iy & a-z \end{pmatrix} = aI + xX + yY + zZI = \begin{pmatrix} 1 & \\ & 1 \end{pmatrix}, X = \begin{pmatrix} & 1 \\ 1 & \end{pmatrix}, Y = \begin{pmatrix} & -i \\ i & \end{pmatrix}, Z = \begin{pmatrix} 1 & \\ & -1 \end{pmatrix}$$

Bloch Bloch

transmon qubit $|0\rangle$ $|1\rangle$

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

E_0 E_1

$$H_0 = E_0|0\rangle\langle 0| + E_1|1\rangle\langle 1| = \begin{pmatrix} E_0 & \\ & E_1 \end{pmatrix} = \begin{pmatrix} \frac{E_0+E_1}{2} - \frac{E_1-E_0}{2} & \\ & \frac{E_0+E_1}{2} + \frac{E_1-E_0}{2} \end{pmatrix} = -\frac{\omega}{2}Z + \frac{E_0+E_1}{2}I$$

$\omega = E_1 - E_0$

$$H_0 = -\frac{\omega}{2}Z$$

$\tilde{U} = e^{-iH_0 t}$

$$|\psi_I\rangle = \tilde{U}|\psi_S\rangle H_I = \tilde{U}^\dagger H_S \tilde{U} + i\frac{\partial \tilde{U}^\dagger}{\partial t}\tilde{U} = e^{iH_0 t} H_0 e^{-iH_0 t} + i\left(\frac{\partial}{\partial t} e^{iH_0 t}\right) e^{-iH_0 t} = H_0 + i \cdot iH_0 \cdot e^{iH_0 t} e^{-iH_0 t} = H_0 - H_0 = 0$$

H_0

“ ”

$$\begin{aligned} H' = \Omega e^{i(\omega_d t - \phi)}|0\rangle\langle 1| + h.c. &= \Omega\left(\left(\cos\left(\omega_d t - \phi\right) + i \sin\left(\omega_d t - \phi\right)\right)|0\rangle\langle 1| + \left(\cos\left(\omega_d t - \phi\right) - i \sin\left(\omega_d t - \phi\right)\right)|1\rangle\langle 0|\right) \\ &= \Omega\left(X \cos\left(\omega_d t - \phi\right) - Y \sin\left(\omega_d t - \phi\right)\right) \end{aligned}$$

$H = H_0 + H'$ $\omega_d = \omega$ (RWA)

$$H_I = \tilde{U}^\dagger H' \tilde{U} = \Omega\left(X \cos\phi + Y \sin\phi\right)$$

$\text{rfUnitary}(\theta, \phi)$

$$\text{rfUnitary}(\theta, \phi) = e^{-i\frac{\theta}{2}(X\cos\phi + Y\sin\phi)}$$

XY- ϕ θ $\theta \propto \Omega t$ Ω t

$\theta = \pi/2$

$$\text{R}(\phi) = \text{rfUnitary}(\pi/2, \phi)$$

$\text{P}(\phi) = |0\rangle\langle 0| + e^{-i\phi}|1\rangle\langle 1|$

2.2

transmon (QND)

ω_r Sig `'trace_avg'`

Sig ω_r IQ S_{21} `iq` shots S_{21} `'iq_avg'`

$|0\rangle$ $|1\rangle$ S_{21} IQ IQ $|0\rangle$ $|1\rangle$ `'state'` shot `cbit` shots

AD `signal` `'remote_'`

3

T_1 T_2

1. NA/ AD S21 $|0\rangle$ 2. 2. 'iq_avg' 'population' 01Spectrum 01 3. 3. 'iq_avg' 'population' 01Ramsey 4 2. 4. 'iq_avg' 'population' 01PowerRabi single shot 5 drive 6 5. 01 Scatter 'population' 3. 6. 'population' 01Delta 4

7. 'Count' 8. 'RB'

[8]:
```
%matplotlib notebook

import numpy as np
from matplotlib import pyplot as plt

import kernel
# kernel.init()
from qos_tools.experiment.scanner2 import Scanner

from itertools import chain
from typing import Optional, Any, Union
from qos_tools.experiment.libs.tools import generate_spanlist
from home.hkxu.tools import get_record_by_id
from waveforms.visualization.widgets import DataPicker

plt.rcParams['xtick.direction'] = 'in'
plt.rcParams['ytick.direction'] = 'in'
plt.rcParams['xtick.top'] = True
plt.rcParams['ytick.right'] = True
plt.rcParams['xtick.minor.visible'] = True
plt.rcParams['ytick.minor.visible'] = True
plt.rcParams['image.origin'] = 'lower'
plt.rcParams['figure.figsize'] = [9, 3]
plt.rcParams['font.size'] = 8
plt.rcParams['lines.linewidth'] = 1
plt.rcParams['lines.markersize'] = 2
plt.rcParams['lines.marker'] = '.'
plt.rcParams['pdf.fonttype'] = 42
plt.rcParams['ps.fonttype'] = 42
plt.rcParams['xtick.labelsize'] = 6
plt.rcParams['ytick.labelsize'] = 6
```

[13]:
```
import time
from functools import lru_cache

default_shots = kernel.get('station.shots')
print(default_shots)
```

```
@lru_cache(maxsize=None)
def init_bias():
    ret = []
    return ret

@lru_cache(maxsize=None)
def fina_bias():
    ret = []
    return ret

def general_run_task(para_dict, timeout, print_task=True, bar=True,␣
 ↪default_shots=default_shots, **kw):

    test = kernel.create_task(Scanner, args=(), kwds=para_dict['init'])
    test.init(**para_dict)
    test.shots = default_shots
    time.sleep(0.1)
    task = kernel.submit(test, **kw)
    if bar:
        task.bar()
    time.sleep(0.1)
    task.join(timeout)
    time.sleep(0.1)
    if print_task:
        print(task)
    return task
```

```
1024
```

3.1 Measure

3.1.1 S21

[26]:

```
def S21(qubits: list[str],
                    center: Optional[Union[float, list[float]]] = None, delta:␣
 ↪Optional[float] = None, st: Optional[float] = None, ed: Optional[float] =␣
 ↪None, mode: str = 'linear', sweep_points: int = 101,
                    signal: str = 'iq_avg', repeat=1, **kw) -> dict:
    """
    [f'Q{i}'] Measure S21 without constraints, change awg frequency.

    Args:
        qubits (list[str]): qubit names.
        center (Optional[Union[float, list[float]]], optional): sweep center.␣
↪Defaults to None.
        delta (Optional[float], optional): sweep span. Defaults to None.
        st (Optional[float], optional): sweep start. Defaults to None.
        ed (Optional[float], optional): sweep end. Defaults to None.
        sweep_points (int, optional): sweep points. Defaults to 101.
        mode (str, optional): sweep mode. Defaults to 'linear'.
        signal (str, optional): signal. Defaults to 'iq_avg'.
    """

    cts = {q: kernel.get(f'gate.Measure.{q}.default_type') for q in qubits}
    cts = {q: 'params' if cts[q]=='default' else cts[q] for q in qubits}

    if center is None:
        center = [kernel.get(
            f'gate.Measure.{q}.{cts[q]}.frequency') for q in qubits]

    elif isinstance(center, float):
        center = [center]*len(qubits)

    sweep_list = generate_spanlist(
        center=0, delta=delta, st=st, ed=ed, sweep_points=sweep_points,␣
↪mode=mode)

    return {
        'init': {
            'name': 'S21',
            'qubits': qubits,
            'signal': signal,
        },
        'setting': {
            'circuit':
            init_bias()+
            [
                ('Barrier', tuple(qubits)),
                (('Delay', 2e-6), qubits[0]),
                ('Barrier', tuple(qubits)),
                *[(('Measure', j), q) for j, q in enumerate(qubits)],
            ]
            +fina_bias()
            ,
        },
        'sweep_config': {
            q: {
                'addr': f'gate.Measure.{q}.{cts[q]}.frequency',
            }
            for q in qubits
        },
        'sweep_setting': {
#               'repeat': np.arange(repeat),
            tuple(qubits): tuple([
                sweep_list + center[j]
                for j, i in enumerate(qubits)
            ]),
        },
    }
```

[27]:
```
qubits = ['Q0', 'Q11']

st, ed, sweep_points, signal = -1e6,1e6, 31, 'remote_iq_avg'

para_dict = S21(qubits=qubits, st=st, ed=ed, sweep_points=sweep_points,␣
 ↪mode='log', signal=signal)
```

[17]:
```
task = general_run_task(para_dict, 1800, dry_run=True, bar=False)
```

```
S21(11093862863540805035, record_id=183814)
```

[25]:
```
task.plot_prog_frame(0, start=0e-6, stop=6.5e-6, raw=True, sample_rate=6e9,␣
 ↪keys=['M0', 'M1', 'Q0', 'Q11'])
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
result = task.result()

fig, ax = plt.subplots((len(qubits)+4)//5, 5, figsize=[8, (len(qubits)+4)//5*1.
 ↪6])
ax = ax.flatten()
fig.suptitle(f"{task.name} id={task.record_id}")

from qos_tools.analyzer.tools import get_normalization, get_convolve_arg

cali = {}

for i, q in enumerate(qubits):
    flag, ans = get_convolve_arg(x=result['index'][q][:], y=np.
 ↪abs(result[signal][:, i]),
                                  ax=ax[i], ext='min')
    if flag:
        ax[i].axvline(x=result['index'][q][np.argmin(get_normalization(np.
 ↪abs(result[signal][:, i])))], c='k', ls='--')
```

```
#         cali[f'gate.Measure.{q}.params.frequency'] = result['index'][q][np.
↪argmin(get_normalization(np.abs(result[signal][:, i])))]
        cali[f'gate.Measure.{q}.params.frequency'] = ans
#     ax[i].plot(result['index'][q][:], (np.abs(result[signal][:, i]))/1e8, '.
↪-')
#     cali[f'gate.Measure.{q}.params.frequency'] = result['index'][q][np.
↪argmin(np.abs(result[signal][:, i]))]
#     ax[i].axvline(x=cali[f'gate.Measure.{q}.params.frequency'], c='r')
    ax[i].set_title(q, fontsize=8)

fig.tight_layout()
fig.show()
```

[ ]:
```
kernel.update_parameters(cali)
plt.close('all')
```

3.2 R

[ ]:

[ ]:

[ ]:

[ ]: