

Contrôler et mesurer un qubit avec une Quantum Machine

- **Points de cours:**
 - Rotations arbitraires sur la sphère de Bloch
 - Modulation IQ
 - Modulation à Bande Latérale Unique (BLU)
- **Modulation IQ & BLU avec la Quantum Machine**
- **Point de cours:**
 - Mesure dispersive d'un qubit supraconducteur
 - **Mesure dans le plan IQ avec la Quantum Machine**

Démarrage

- Lancez Spyder, vérifiez que le kernel Python est en 3.12
- Choisissez Qt5 pour la sortie graphique dans les préférences, ou exécutez %matplotlib qt5 dans la console
- Le répertoire de travail est \Desktop\TP Quantum Machine
- Le répertoire \TP Quantum Machine\tpqm sera régulièrement mis à jour, pour cela vous devrez faire un clic droit, git bash puis exécuter « git pull » dans la console
- Vous devez copier les fichiers de ce répertoire vers votre propre répertoire, ne travaillez pas directement dans \tpqm
- Vérifier dans un browser que vous voyez la QM à l'adresse 192.168.88.251 ou 192.168.88.253. Notez le nom du cluster, l'adresse et le nom du cluster doivent être copiés dans le fichier « qm_ip.py ».
- Depuis Spyder, exéutez configuration.py. Vous devriez voir un message de connection à la QM.
- **Pensez à régulièrement sauver des données, figures,...**

Hello QUA

```
from qm.qua import *
from configuration import qm

#####
# The QUA program #
#####
with program() as prog:
    with infinite_loop_():
        play("pulse", "rf1")

#####
# Execute the program #
#####
job = qm.execute(prog)
```

Chargement de la configuration et ouverture de la QM

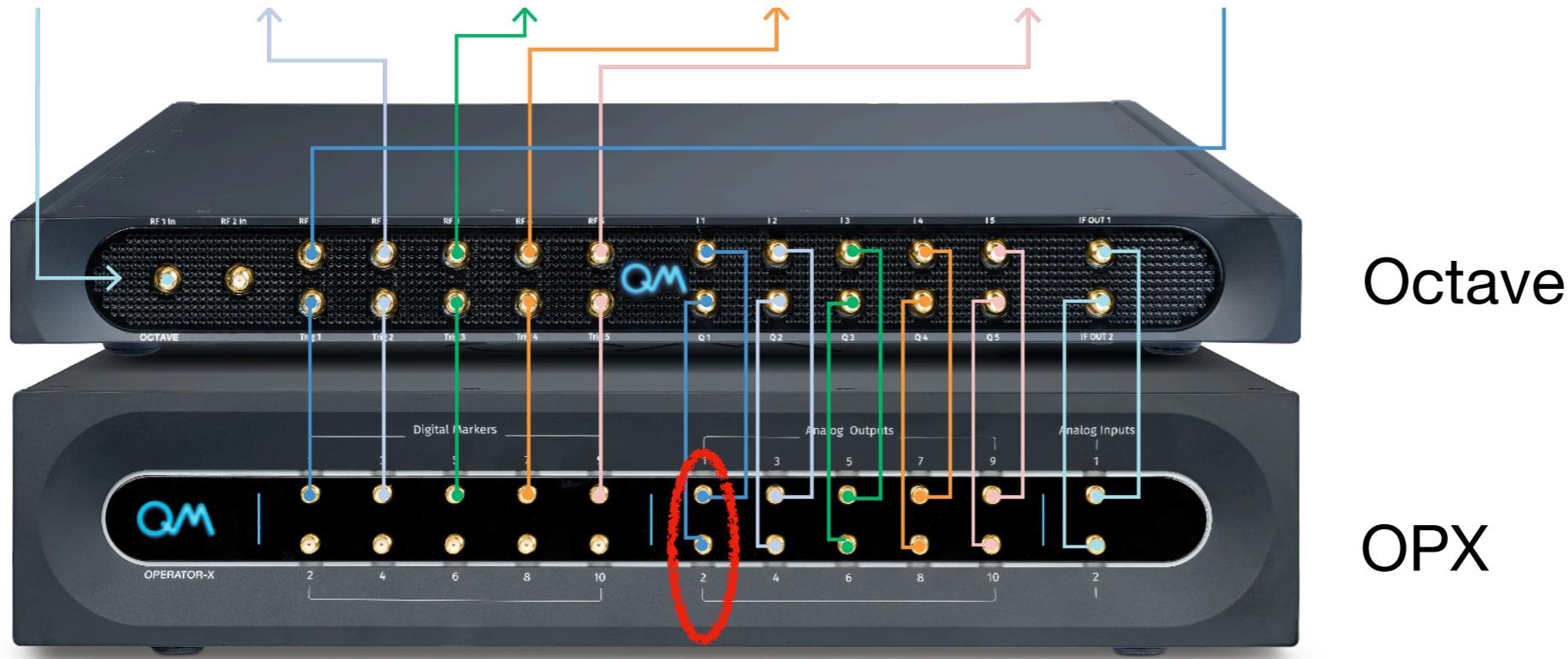
Définition du programme

Lance l'exécution du programme sur la QM

Pour arrêter le code: job.halt() dans la console

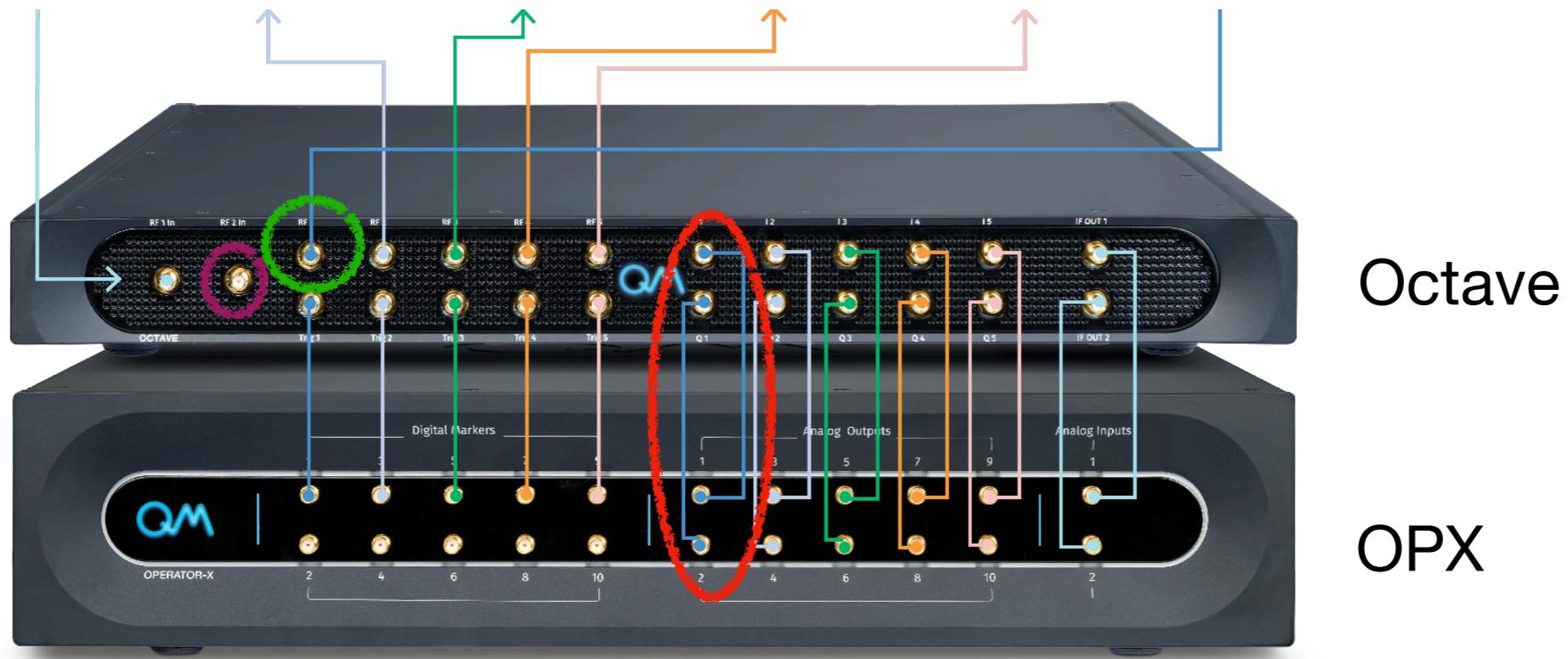
Lien vers la doc: <https://docs.quantum-machines.co/latest/>

Visualiser les signaux de l'OPX



- Branchez les sorties analogiques 1&2 de l'OPX sur l'oscilloscope
- Exécutez le programme « 01_play_pulse », modifiez la fréquence et l'amplitude en modifiant « configuration.py »
 - Modifiez le programme pour obtenir deux pulses séparés de 1µs
 - Modifiez le programme pour obtenir deux pulses séparés de 1µs avec un déphasage de $\pi/2$ entre les pulses (Hint: utiliser amp() avec quatre paramètres)

Visualiser les signaux de l'Octave



- Brancher les sorties analogiques 1&2 de l'OPX sur l'Octave
- Brancher la sortie RF1 sur un spectro ou utiliser la QM en spectro
- Configurez et exédez le programme « 01_play_pulse » pour avoir un pulse à 2.52 GHz avec un LO à 2.5 GHz
- *Pour utiliser la QM en spectro: branchez la sortie RF1 sur RF2 in et utilisez le programme « 03_spectro ». Les paramètres de config sont dans « configuration_spectro.py »*

Calibration des mixers IQ

- Evaluatez les puissances relatives entre les différents signaux visibles sur le spectre
- Appliquez la procédure de calibration « calibrate_mixer.py »
- Observez à nouveau le spectre
- Changez les offsets DC sur les mixers: dans la console exécuter:
`qm.set_output_dc_offset_by_element('rf1','Q',-0.01)` et essayez avec d'autres valeurs

Démodulation avec la QM

```
with program() as prog:
    I = declare(fixed) # QUA variable for the measured 'I' quadrature
    Q = declare(fixed) # QUA variable for the measured 'Q' quadrature
    I_st = declare_stream() # Stream for the 'I' quadrature
    Q_st = declare_stream() # Stream for the 'Q' quadrature

    with infinite_loop_():
        play("pulse", "rf1")

    with infinite_loop_():
        # Demodulate the signals to get the 'I' & 'Q' quadratures)
        measure(
            "readout",
            "spectro",
            None,
            dual_demod.full("cos", "sin", I),
            dual_demod.full("minus_sin", "cos", Q),
        )
        # Save the 'I' & 'Q' quadratures to their respective streams
        save(I, I_st)
        save(Q, Q_st)

    with stream_processing():
        # Cast the data into a 1D vector, and store the results on the OPX processor
        I_st.buffer(n_points).zip(Q_st.buffer(n_points)).save(`IQ`)
```

Live plot des résultats

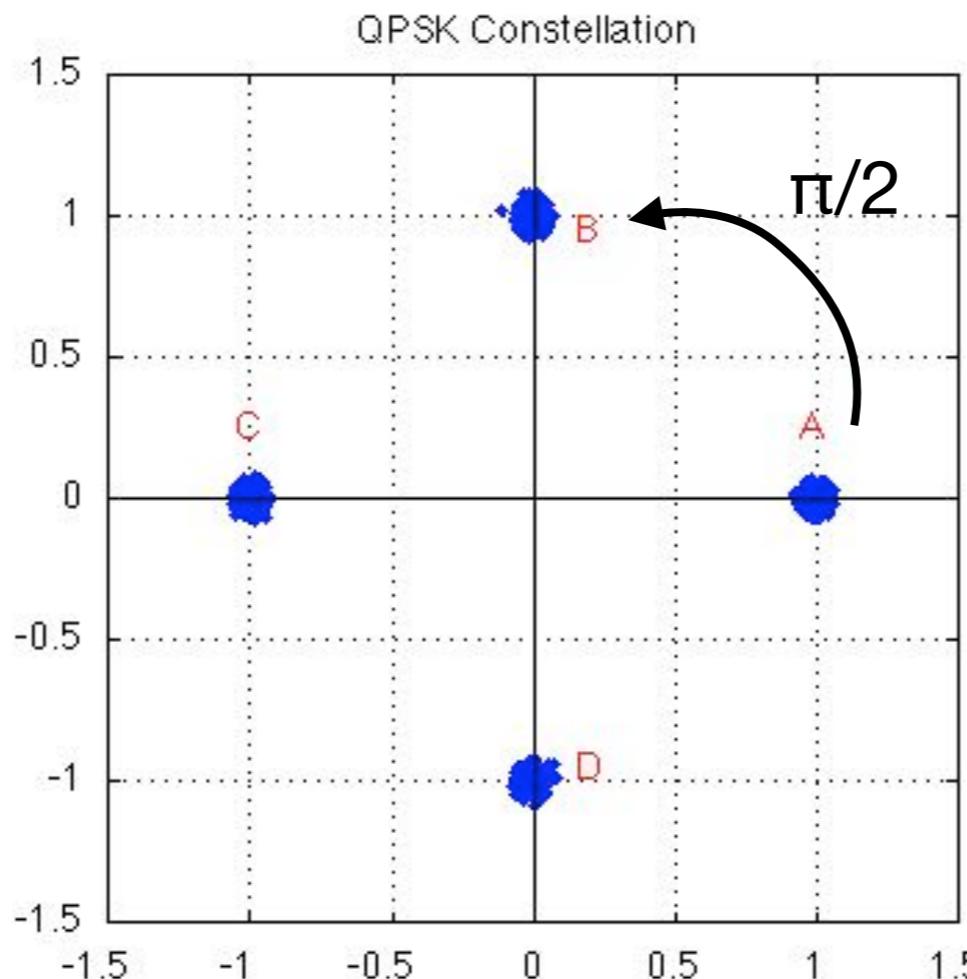
```
#####
# Live plot      #
#####
class myLivePlot(LivePlotWindow):
    def create_axes(self):
        # Create plot axes
        self.ax = self.canvas.figure.subplots()
        # Plot
        self.spectrum = self.ax.plot((spectro_L0 + frequencies)/1e6,np.ones(len(frequencies)))[0]
        self.ax.set_xlabel('Frequency (MHz)')
        self.ax.set_ylabel('Signal (dB)')
        self.ax.set_ylim(-120,-10)

    def polldata(self):
        # Fetch the raw ADC traces and convert them into Volts
        IQ = self.job.result_handles.get("IQ").fetch(1)
        if IQ is None:
            return
        I = IQ['value_0']
        Q = IQ['value_1']
        self.spectrum.set_ydata(10*np.log10(I**2+Q**2))
        self.canvas.draw()

#####
# Execute the program #
#####
job = qm.execute(prog)
qm.set_io1_value(1.0)
window = myLivePlot(job)
window.show()
```

Communication par modulation IQ

- Ecrivez un programme qui envoie en continu des pulses de $77\mu\text{s}$ dont la phase tourne de $\pi/2$ à chaque pulse sur RF1



- Démodulez sur RF2 pendant $10 \mu\text{s}$ et observer le résultat (faire des paquets de 1024 points)

Communication par modulation IQ

- Changez la durée des pulses à 20 μ s et utilisez le code suivant pour l'émission

```
msg=np.load("msg_1007.npz")
msgI=msg["x"]
msgQ=msg["y"]

with program() as prog:
    n = declare(int) # QUA variable for the averaging loop
    m = declare(int)
    Im = declare(fixed,value=msgI)
    Qm = declare(fixed,value=msgQ)
    I = declare(fixed) # QUA variable for the measured 'I' quadrature
    Q = declare(fixed) # QUA variable for the measured 'Q' quadrature
    I_st = declare_stream() # Stream for the 'I' quadrature
    Q_st = declare_stream() # Stream for the 'Q' quadrature

    with infinite_loop_():
        with for_(m, 0, m < len(msgI), m + 1):
            play("pulse" * amp(Im[m],0.,0.,Qm[m]), "rf1")
```

- Démodulez sur RF2 pendant 10 μ s et observez le résultat
- Reprenez la manip avec les antennes
- Envoyez un message à vos voisins