IQ mixer tuning

Ding Ruiqi

January 2021

1 Introduction

The pairs of QM analog ports: (1, 2), (3, 4)... generates In phase (I) and Quadrature (Q) signals at lower frequency (e.g. ω =50 MHz) which can be mixed with the Local oscillator (LO) signal at higher frequency (e.g. Ω =6 GHz):

$$\underbrace{\sin\omega t\ \cos\phi}_{\text{In phase}} + \underbrace{\cos\omega t\ \sin\phi}_{\text{Quadrature}} = \sin(\omega t + \phi) \tag{1}$$

By adjusting the amplitudes $\cos\phi$ and $\sin\phi$ of I and Q respectively, we can adjust the phase of the resulting RHS signal.

The LO and IQ signals are combined to produce Lower Side Band (LSB) and Higher Side Band (HSB) signals:

$$\sin(\Omega t + \Phi)\cdot\sin(\omega t + \phi) = \frac{1}{2}\cos[\underbrace{(\Omega-\omega)}_{\text{LSB}}t + (\Phi - \phi)] \tag{2}$$

$$-\frac{1}{2}\cos[\underbrace{(\Omega+\omega)}_{\text{HSB}}t + (\Phi + \phi)] \tag{3}$$

The actual IQ mixer device does follow the above simple equations as the amplitudes of LSB and HSB can be adjusted via the IQ signal.

2 Introducing the equipment

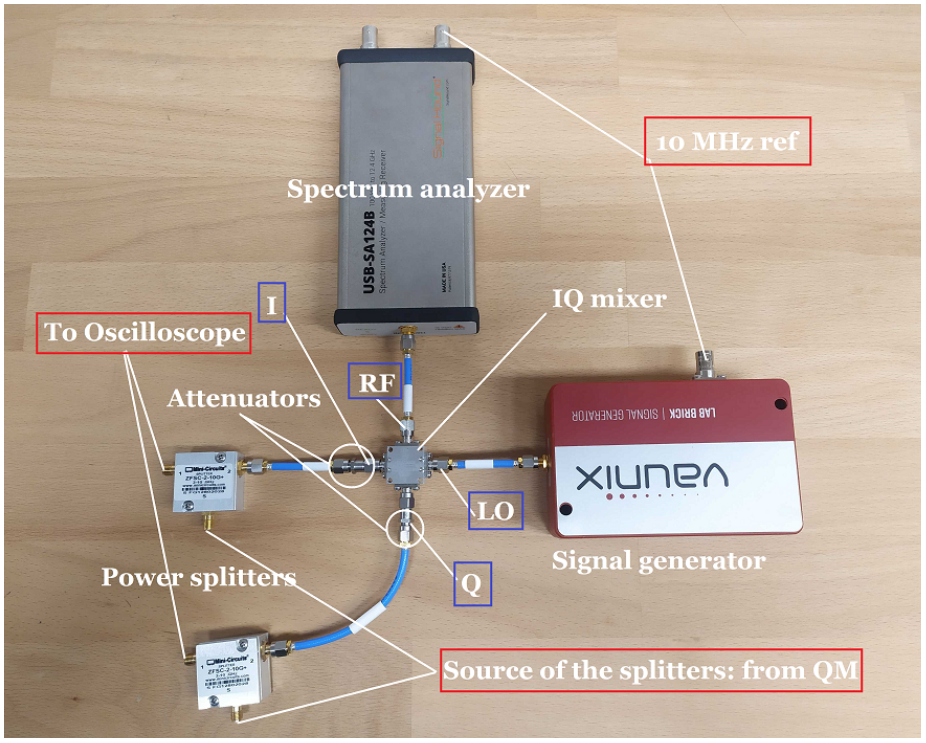


Figure 1: Mixer tuning setup. Texts without box labels the physical equipment. Texts with blue box label the ports of the IQ mixer. Texts with red box label the external connections which are too large to show in the photo.

IQ signals from **QM** analog output ports are sent to two **power splitters**. The splitters are *not necessary* but enables us to observe and debug the IQ output from the **oscilloscope**. The other outputs of the splitters are then fed to the **IQ mixer** through 10 dB **attenuators**. The attenuators are *necessary* to satisfy the input power requirement of the IQ mixer. The IQ signals are combined with the LO signal from the **signal generator** to produce an output on the RF port, which is analyzed by the **spectrum analyzer**.

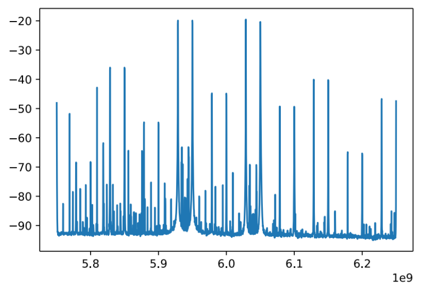
3 Setup the equipment correctly

3.1 Signal generator

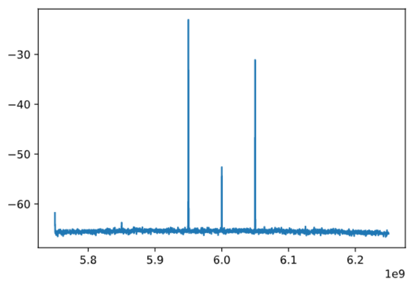
- The signal generator must be connected to and external reference clock of 10 MHz. And declared in the python program. This is already done by Ruiqi's library.
- The output frequency and power must be checked and reset manually upon **restarting** the device.
- The Pll mode should be turned on.

3.2 Spectrum analyzer

- The spectrum analyzer must be connected to and external reference clock of 10 MHz. And declared in the python program. This is already done by Ruiqi's library.
- The image rejection option should be set to *true* for continuous measurement. And *false* for pulses measurement. It is set to *true* in Ruiqi's library. The consequences of not using image rejection is the appearance of a series of faulty peaks in the spectrum, as shown in Figure 2.



(a) before image rejection



(b) after image rejection

Figure 2: Connecting the signal generator directly to the spectrum analyzer (set to 6 GHz). **(a)** when image rejection is set to *false*. **(b)** when image rejection is set to *true*.

- The measurement code. In any measurement one should first initialize the device by specifying the **center, span, number of points, estimated power** of the sweep, this step is time consuming so it should only be executed **once**. Subsequent measurement only gets the sweep data without setting these values.
- The above step measures a frequency range determined by **center, span**. In order to measure the power at a specific frequency, on should set **span**

= **250 MHz** and **number of points** = **1**. That is, a single frequency measurement is simply a spectrum measurement with special parameters.

- The code for the above are provided by 4 functions in Ruiqi's code. The source code can be found at https://github.com/tesla-cat/LabTools/blob/master/LabControl/Instruments/SA124B/SA124B.py, the application example can be found at https://github.com/tesla-cat/LabTools/blob/master/LabControl/tutorial-2-2-SA124B-MixerTuning.ipynb

```
def getFreqsAndAmps(self, val):
    ...
def initSweep(self, center, span, N, power):
    ...
def initSweepSingle(self, center, power):
    return self.initSweep(center, span = 250e3, N = 1, power)
def getSweep(self, length):
    ...
def getSweepSingle(self, length):
    return self.getSweep(length)[length // 2]
```

3.3 QM

- Sometimes one may observe unexpected waveform on the QM output as shown in Figure 3. (corresponding to new frequencies components in the spectrum). The appearance of these components are likely caused by an voltage overflow. The output range of QM machine is $[-0.5, 0.5]$ V. Applying the IQ imbalance matrix for example can result in higher voltage values at the analog output than the voltage specified on the **waveforms** in the **configuration.py** file. One need to make sure that the overall voltage is within the range of the analog outputs. Any overflow will be automatically clamped by QM therefore causing unwanted harmonics.

- To resolve this issue one can simply adjust the amplitude in the **configuration.py**.

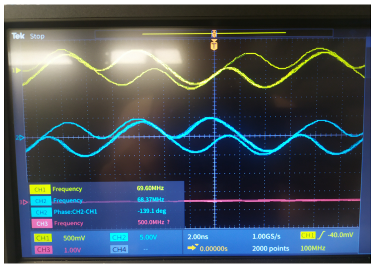


Figure 3: Unexpected waveform on the QM output (corresponding to new frequencies components in the spectrum)

4 Tune the mixer

4.1 A general optimization algorithm

We have two steps for mixer tuning: Carrier(LO) Leakage and IQ Imbalance. Despite the difference in physical nature, their solution reduces to a simple two parameter minimization problem.

Algorithm 1: A simple general 2D minimization algorithm for mixer tuning

```
Result: xmin and ymin that minimizes the amplitude of certain
        frequency f in the spectrum
set the initial center and span of x and y;
for some iterations do
    for points on a 2D grid of x and y do
        set the QM machine based on x, y. Handled by a callback
          function in the actual code below;
        sample the amplitude of f at this point;
    end
    find the xmin, ymin that gives the minimum amplitude;
    update the centers to xmin, ymin;
    reduce the span by some ratio;
end
```

```python
def minimize(self, centers, spans, callback, amps, numIters, numGrids, shrink):
    spans = np.array(spans)
    def minimizeIter(centers_, spans_):
        x = np.linspace(centers_[0]-spans_[0]/2, centers_[0]+spans_[0]/2, numGrids[0])
        y = np.linspace(centers_[1]-spans_[1]/2, centers_[1]+spans_[1]/2, numGrids[1])
        power = np.zeros(numGrids)
        for i, xi in enumerate(x):
            for j, yj in enumerate(y):
                callback(xi, yj)
                power[i, j] = self.sa124B.getSweepSingle(len(amps))
        if numGrids[0] == 1:
            plt.plot(y, power[0, :])
            plt.show()
        elif numGrids[1] == 1:
            plt.plot(x, power[:, 0])
            plt.show()
        else:
            fig = go.Figure( data=[go.Surface(z=power, x=x, y=y)] )
            fig.show()
        ind = np.unravel_index(np.argmin(power, axis=None), power.shape)
        minx, miny = x[ind[0]], y[ind[1]]
        return minx, miny
    for i in range(numIters):
        print('iter, centers, spans:', i, centers, spans)
        centers = minimizeIter(centers, spans)
        spans = spans * shrink**(i+1)
    print('final centers:', centers)
    return centers
```

The full python code can be found from https://github.com/tesla-cat/Flask-React-Lab-Control/blob/master/LabControl/tutorial-2-2-SA124B-MixerTuning.ipynb.

4.2 Carrier(LO) Leakage

Carrier(LO) Leakage is caused by DC-offsets in both I and Q ports of the mixer. This offset is caused by a conversion loss imbalance between the two mixers which are inside the IQ mixer shown in Figure 4.

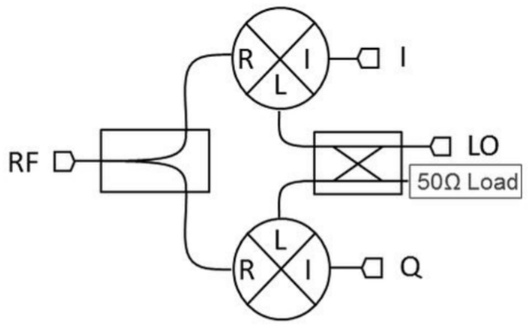


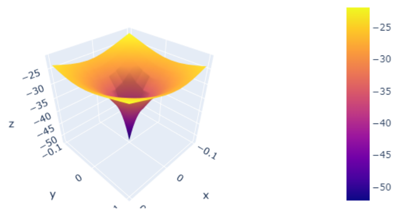
Figure 4: Internal structure of the IQ mixer consists of two simple mixers.

The main consequence is that when performing up-conversion with the IQ mixer a certain amount of the LO signal will leak into the mixer RF output. The LO to RF isolation, one of the metrics of an IQ mixer, quantifies this carrier leakage.

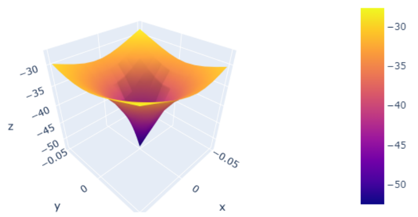
To address this problem, an external DC voltage input is applied in the I and Q ports in a way that the LO leakage is minimal, this corresponds to the following python code:

```
def step2CarrierLeakage(self, centers, spans, numIters, numGrids, shrink):
    print('step2CarrierLeakage')
    _, amps = self.sa124B.initSweepSingle(center=self.carrier, power=self.power)
    def callback(I, Q):
        self.qm.set_dc_offset_by_qe("qubit", "I", float(I))
        self.qm.set_dc_offset_by_qe("qubit", "Q", float(Q))
    return self.minimize(centers, spans, callback, amps, numIters, numGrids, shrink)
```

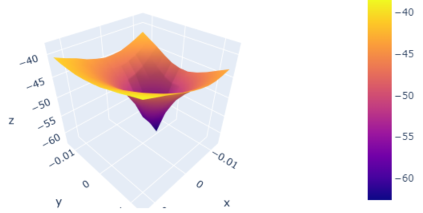
The minimal values are found in a few iterations over the search grid, resulting in the landscapes shown in Figure 5.



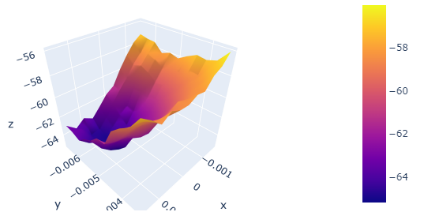
(a) 1st minimization iteration (MI)



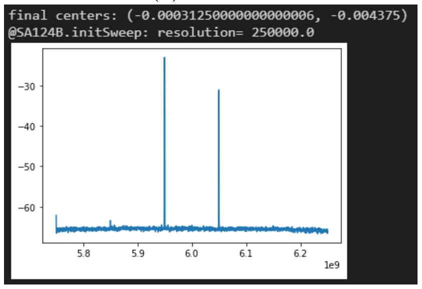
(b) 2nd MI



(c) 3rd MI



(d) 4th MI



```
final centers: (-0.00031250000000000006, -0.004375)
@SA124B.initSweep: resolution= 250000.0
```

(e) resulting spectrum

Figure 5: **(a)-(d)** Four iterations of minimization of the LO leakage, x and y are IQ DC offsets, and z is the amplitude of LO frequency. **(e)** The resulting spectrum, LO leakage at 6 GHz is perfectly eliminated.

4.3 IQ Imbalance

IQ Imbalance refers to the amplitude and phase imbalances.

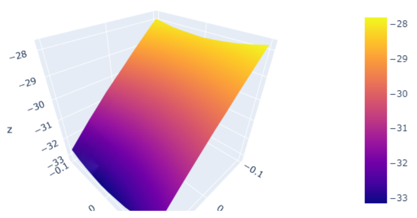
- Amplitude imbalances are caused by unbalances in the quadrature hybrid coupler and different conversion losses in each of the mixers. It means that the amplification / attenuation through the IQ mixer of each of the signals is not identical. The amplitude deviation of a mixer is the metric which quantifies the amplitude imbalance.

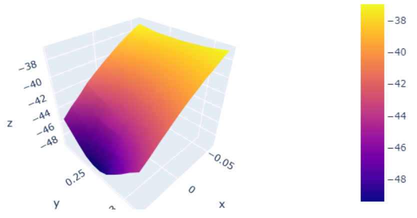
- Phase imbalances are due to phase unbalances of the hybrid coupler and different electrical connection lengths. This imbalance is indicated by the quadrature phase deviation of the mixer. Because of both of these imbalances the cancellation of the unwanted sideband will not be perfect.

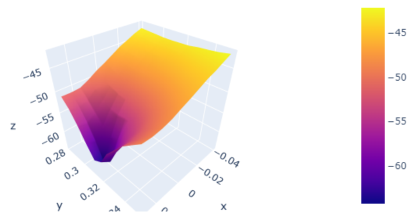
To address this problem, slight modifications in the amplitude and phase of the oscillating signal applied in Q are performed.

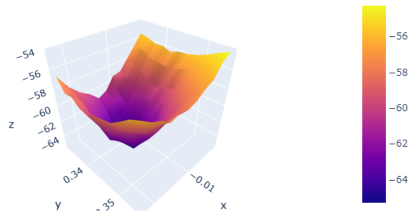
```python
def step3IqImbalance(self, centers, spans, numIters, numGrids, shrink):
    print('step3IqImbalance')
    _, amps = self.sa124B.initSweepSingle(center=self.carrier+self.sideBand,
        power=self.power)
    def callback(gain, phase):
        self.qm.set_mixer_correction("mixer_qubit", qubit_IF, qubit_LO,
            IQ_imbalance(gain, phase))
    return self.minimize(centers, spans, callback, amps, numIters, numGrids, shrink)
```

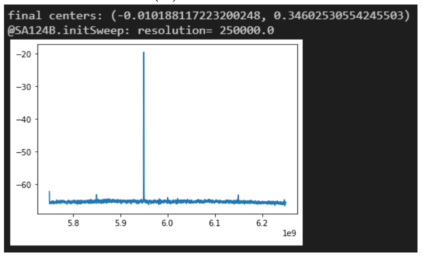
The minimal values are found in a few iterations over the search grid, resulting in the landscapes shown in Figure 6.


(a) 1st minimization iteration (MI)


(b) 4th MI


(c) 5th MI


(d) 7th MI


```
final centers: (-0.010188117223200248, 0.34602530554245503)
@SA124B.initSweep: resolution= 250000.0
```

(e) resulting spectrum

Figure 6: **(a)-(d)** Four selected iterations of minimization of the LO leakage, x and y are **gain** and **phase**, and z is the amplitude of **upper side band** frequency. **(e)** The resulting spectrum, upper side band at 6 GHz + 50 MHz is perfectly eliminated.

5 Results

Each pairs of the QM analog output are tuned via the 3 step tuning process and their optimal values are found as follows:

```
- channels 1, 2: cw = 0.3
    - IQ final centers: (0.015937499999999999, -0.005312500000000001)
    - gain, phase final centers: (-0.025000016106655396, 0.3386585083616496)
- channels 3, 4: cw = 0.3
    - IQ final centers: (0.009999999999999995, -0.008750000000000003)
    - gain, phase final centers: (0.02153621448045734, 0.35205362852657557)
- channels 5, 6: cw = 0.2
    - IQ final centers: (0.008124999999999995, -0.022812500000000003)
    - gain, phase final centers: (0.18426518494587707, 0.39080206877419243)
- channels 7, 8:
    - IQ final centers: (0.0037500000000000007, -0.006562500000000003)
    - gain, phase final centers: (0.09298324113900173, 0.35795467851316987)
- channels 9, 10:
    - IQ final centers: (-0.00031250000000000006, -0.004375)
    - gain, phase final centers: (-0.010188117223200248, 0.34602530554245503)
```