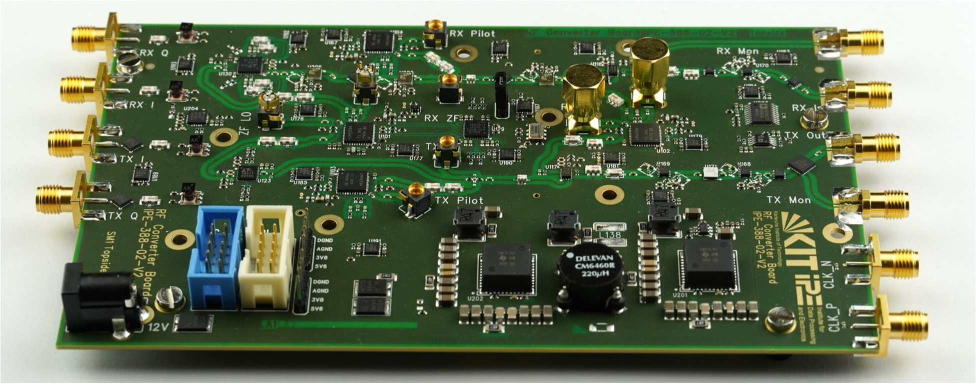
Introduction to QiCode

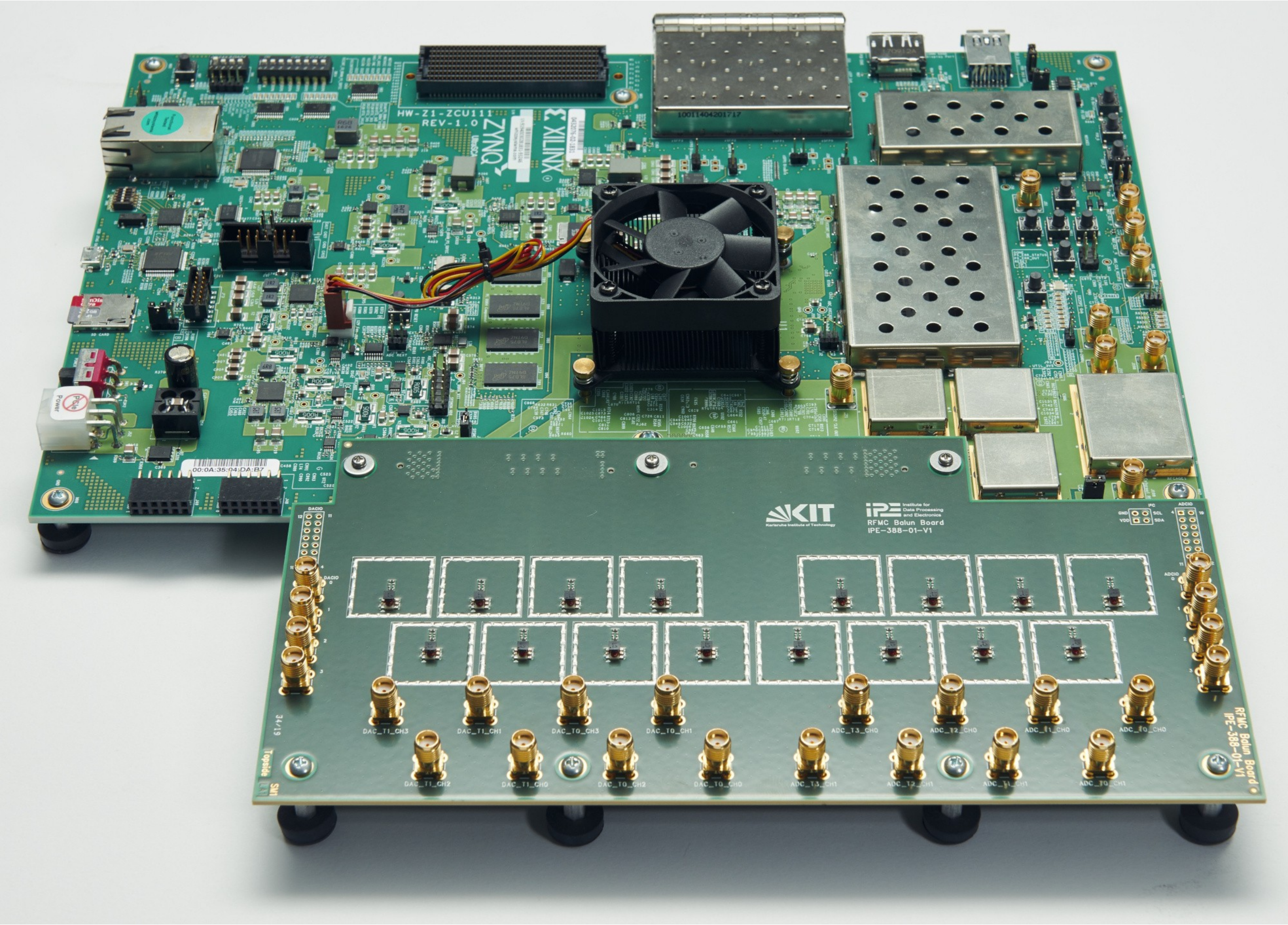
Marvin Fuchs and Robert Gartmann (KIT)

Electronics for Cryogenic Applications

RF-System-on-Chip (RFSoC)
- FPGA
- 2x Processing Unit
- 8x 14 bit DACs with 4 GS/s
- 8x 12 bit ADCs with 4 GS/s



Custom HF-Frontend
- Mixers for frequency conversion



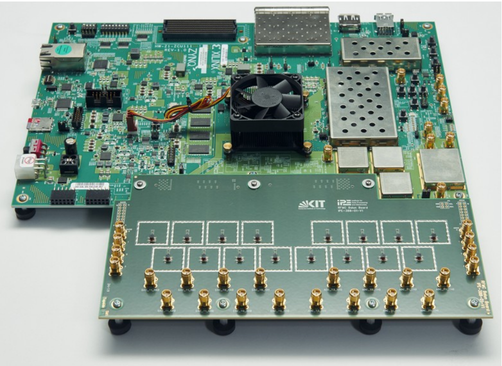
QiController: RFSoC Based Control Platform

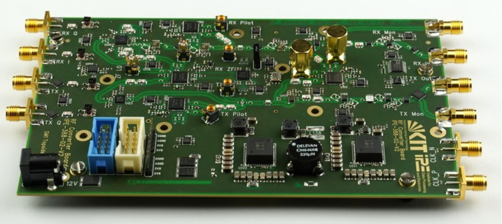
Karlsruhe Institute of Technology

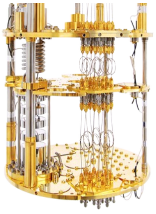
Custom Firmware and Software

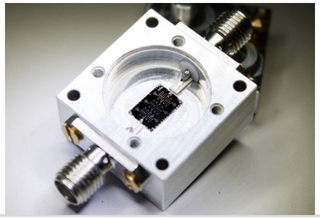
- Python-Client (QiCode) on User PC
- Ethernet for communication with the platform
- ServiceHub as central control instance
- Taskrunner to execute real-time tasks
- One Unit Cell per Qubit

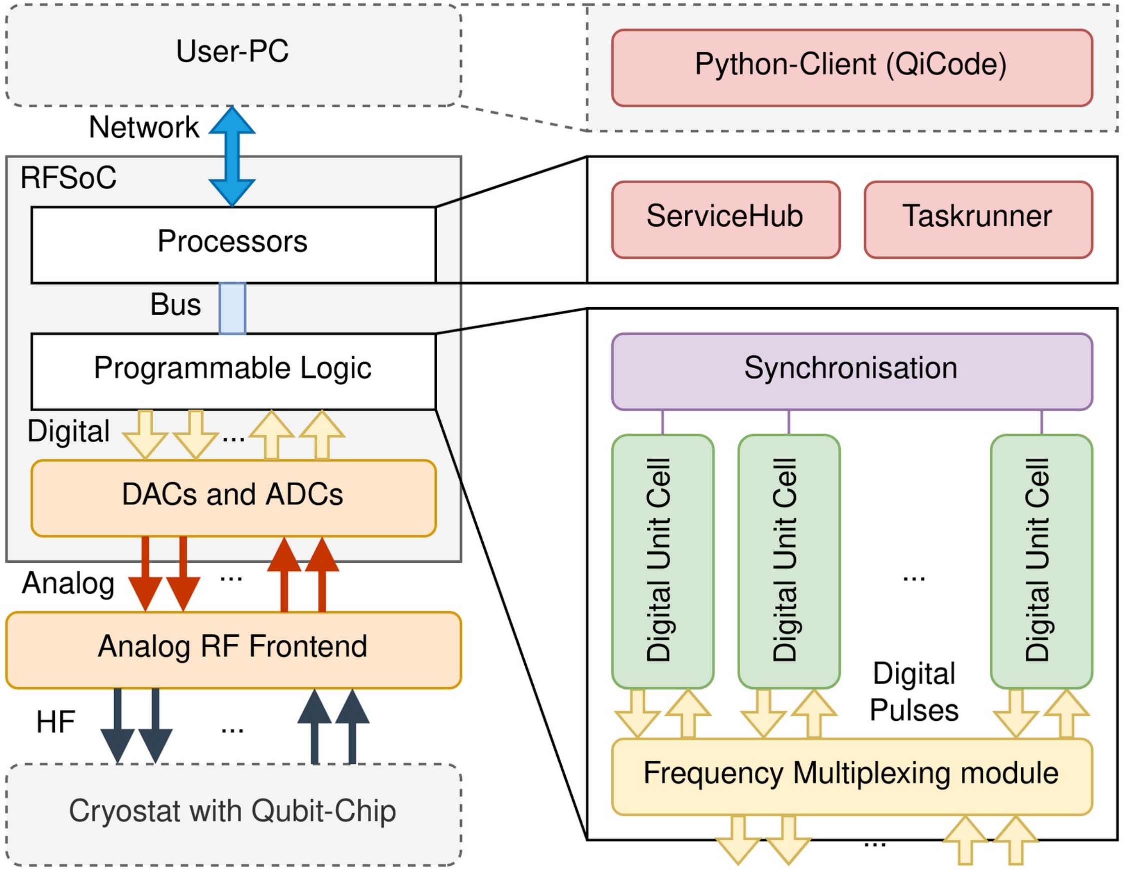
Qiskit qKIT







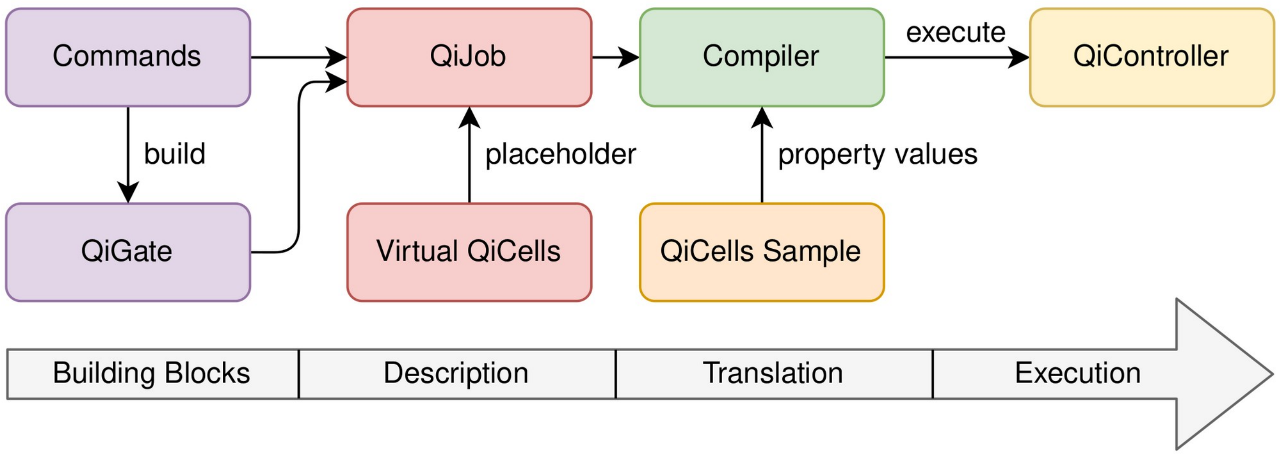


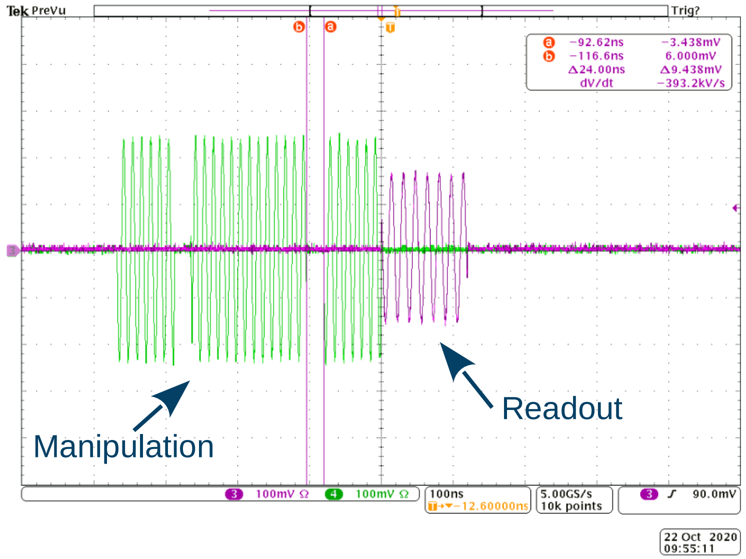


QiCode

- Experiment description language to interface the QiController
- Based on Python

- Usage similar to Qiskit; closer to the sample
- Translation layer to Qiskit





SpinEcho pulses output by the QiController

QiCode and Qiskit

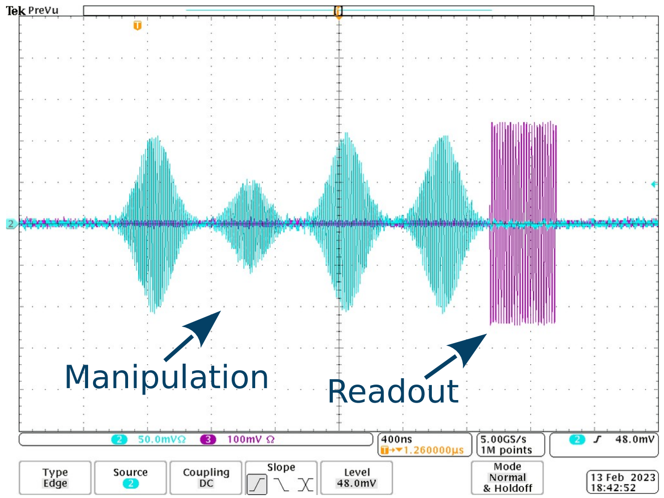
Qiskit

```python
qc = QuantumCircuit(1,1)

qc.x(0)
qc.h(0)
qc.z(0)
qc.y(0)

qc.measure(0,0)
```

QiCode

```python
with QiJob() as job:
    q = QiCells(1)

    # X-Gate
    Play(q[0], QiPulse( length=q[0]["pi"],
                        shape=ShapeLib.gauss,
                        phase=0.0,
                        frequency=q[0]["manip_frequency"]))

    # H-Gate (Ry-Gate and X-Gate)
    Play(q[0], QiPulse( length=q[0]["pi"],
                        shape=ShapeLib.gauss,
                        phase=np.pi / 2,
                        frequency=q[0]["manip_frequency"],
                        amplitude=1 / 2))
    Play(q[0], QiPulse( length=q[0]["pi"],
                        shape=ShapeLib.gauss,
                        phase=0.0,
                        frequency=q[0]["manip_frequency"]))

    # Z-Gate
    RotateFrame(q[0], angle=np.pi)

    # Y-Gate
    Play(q[0], QiPulse( length=q[0]["pi"],
                        shape=ShapeLib.gauss,
                        phase=np.pi / 2,
                        frequency=q[0]["manip_frequency"]))

    # Measure
    PlayReadout(q[0], QiPulse(  length=q[0]["rec_pulse"],
                                frequency=q[0]["rec_frequency"]))
    Recording(  q[0],
                duration=q[0]["rec_length"],
                offset=q[0]["rec_offset"],
                save_to="result")

job.run(qic, sample)
```

Pulses



This level of detail is possible in QiCode, but gates can also be encapsulated as **QiGates**

QiCode and Qiskit

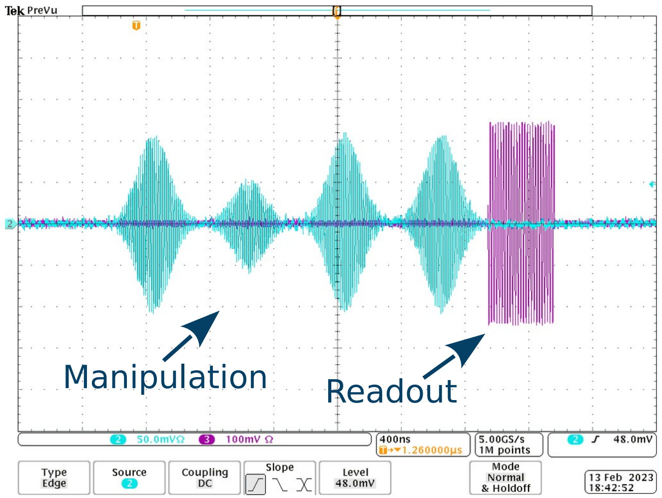
Qiskit

```
qc = QuantumCircuit(1,1)

qc.x(0)
qc.h(0)
qc.z(0)
qc.y(0)

qc.measure(0,0)
```

QiCode

```
with QiJob() as job:
    q = QiCells(1)

    X_Gate(q[0])
    H_Gate(q[0])
    Z_Gate(q[0])
    Y_Gate(q[0])

    Measure(q[0], save_to="result")

job.run(qic, sample)
```

Pulses

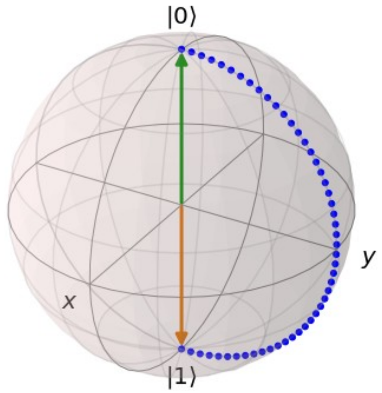


Use Case: Rabi Oscillations

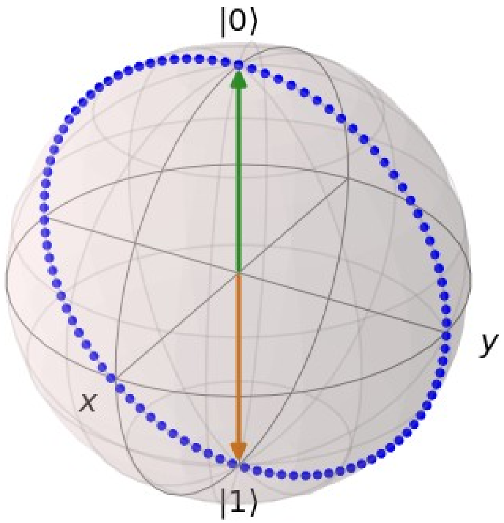
- Oscillations between ground state $|0\rangle$ and excited state $|1\rangle$ due to a driving field
- Used to determine pi-pulse length

What do we need?

- Manipulation pulses of varying length
- Readout of the qubit state



π-Pulse



Rabi Oscillations

Use Case: Rabi Oscillations

```python
@QiGate
def Readout(cell: QiCell, save_to: str = None):
    PlayReadout(cell, QiPulse(length=cell["rec_pulse"],
                              frequency=cell["rec_frequency"]))
    Recording( cell,
               duration=cell["rec_length"],
               offset=cell["rec_offset"],
               save_to=save_to)

@QiGate
def Thermalize(cell: QiCell):
    Wait(cell, delay=5 * cell["T1"])
```

QiSample

```python
sample = QiSample(1) # 1 cell/qubit only
sample[0]["rec_pulse"] = 416e-9 # s readout pulse length
sample[0]["rec_length"] = 400e-9 # s recording window size
sample[0]["rec_frequency"] = 60e6 # Hz readout pulse frequency
sample[0]["manip_frequency"] = 80e6 # Hz control pulse frequency
```

QiGate

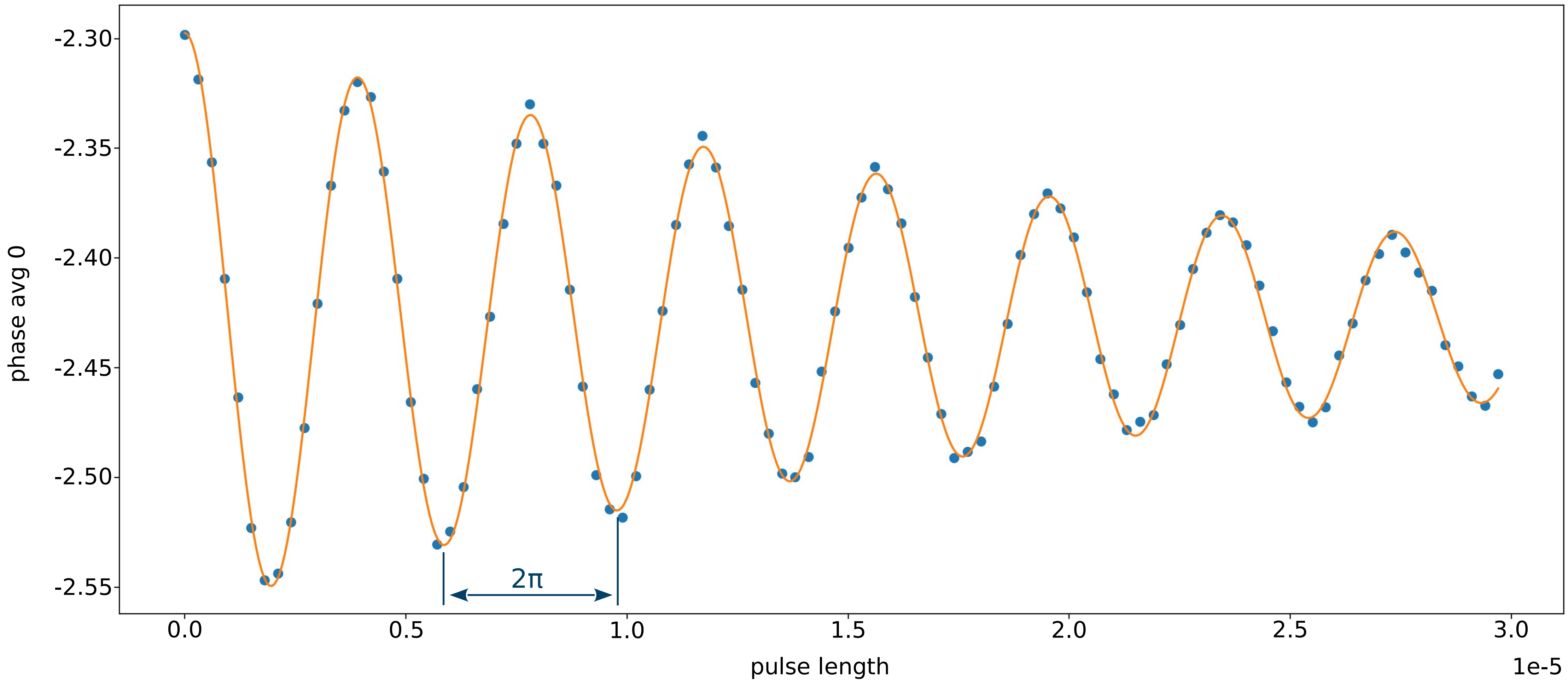
QiSample

```python
with QiJob() as rabi:
    q = QiCells(1)
    length = QiTimeVariable()
    with ForRange(length, 0, 1e-6, 20e-9):
        Play(q[0], QiPulse(length, frequency=q[0]["manip_frequency"]))
        Readout(q[0], save_to="result")
        Thermalize(q[0]) # Wait for the qubit to thermalize

rabi.run(qic, sample, averages=1000)
```

Manipulation pulses of varying length

Readout of the qubit state

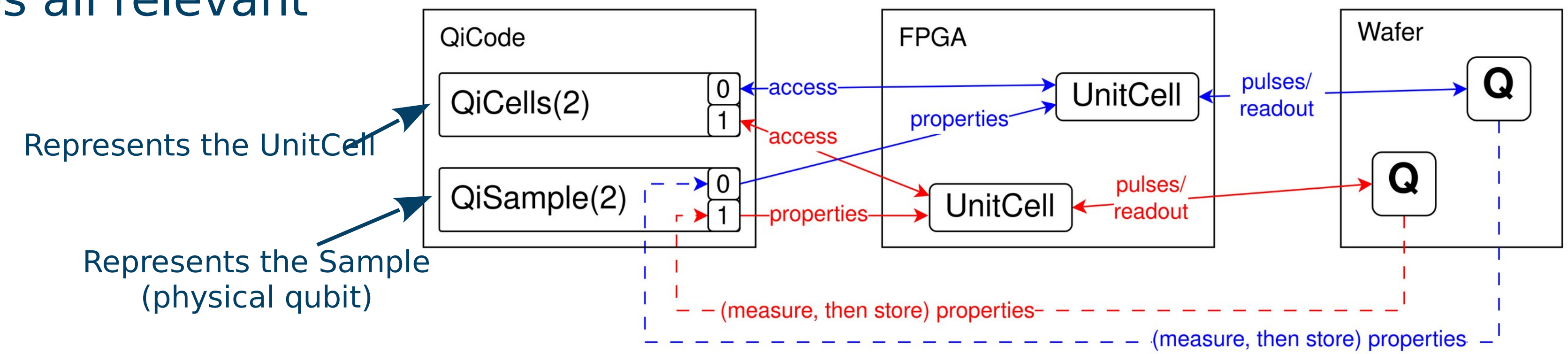
Use Case: Rabi Oscillations



QiSample

- The physical properties of a sample can be stored in an instance of **QiSample**
- A sample can consist of one or more cells
- Each cell corresponds to a qubit and defines all relevant properties

```
sample = QiSample(1) # 1 cell/qubit only
sample[0]["rec_pulse"] = 416e-9 # s readout pulse length
sample[0]["rec_length"] = 400e-9 # s recording window size
sample[0]["rec_frequency"] = 60e6 # Hz readout pulse frequency
sample[0]["manip_frequency"] = 80e6 # Hz control pulse frequency
```



QiGate

Karlsruhe Institute of Technology

- Reusable building blocks can be defined with the decorator **QiGate**
- Equivalent to functions in programming languages
- Can use the physical properties of the cells stored in a **QiSample**

```python
@QiGate
def Readout(cell: QiCell, save_to: str = None):
    PlayReadout(cell, QiPulse(length=cell["rec_pulse"],
                              frequency=cell["rec_frequency"]))
    Recording(
        cell,
        duration=cell["rec_length"],
        offset=cell["rec_offset"],
        save_to=save_to
    )

@QiGate
def PiPulse(cell: QiCell):
    Play(cell, QiPulse(length=cell["pi"],
               frequency=cell["manip_frequency"]))

@QiGate
def Thermalize(cell: QiCell):
    Wait(cell, delay=5 * cell["T1"])
```

QiJob

- Entire experiments can be described in a generic way as instances of **QiJob**
- Can use the concepts of **QiSample** and **QiGate** for abstraction
- Can be created to be easily reusable for different samples

```
# Commands are always encapsulated within the QiJob context
with QiJob() as rabi:
    # First, we define how many qubits the experiment requires
    q = QiCells(1)
    # Rabi consists of variable length excitation pulses,
    # so we need to create a time variable
    length = QiTimeVariable()
    # The variable can then be changed within a for loop
    with ForRange(length, 0, 1e-6, 20e-9):
        # Output the manipulation pulse with variable length
        Play(q[0], QiPulse(length, frequency=q[0]["manip_frequency"]))
        # Perform a consecutive readout (using the above QiGate)
        # The data can later by accessed via the specified name "result"
        Readout(q[0], save_to="result")
        # Wait for the qubit to thermalize (also a QiGate)
        Thermalize(q[0])
```

- QiJobs can be executed

```
rabi.run(qic, sample, averages=1000)
data = rabi.cells[0].data("result")
```

Basics (Commands)

- **`Play(cell, pulse)`**
  - Play the given pulse at the manipulation output for the given cell.
- **`PlayReadout(cell, pulse)`**
  - Same as Play but outputs readout pulses.
- **`Recording(cell, duration, offset, save_to, state_to)`**
  - Records the input of the cell with given duration and offset (e.g. to compensate electrical delay). Typically used directly after a **PlayReadout** command. Using the save_to argument, the result data can be stored and accessed later using the given string. With state_to, the obtained qubit state can be saved to a **QiVariable**.
- **`Wait(cell, delay)`**
  - The specified cell waits for the specified delay before continuing with the next command.

Basics (Parameters)

- **`QiVariable(type)`, `QiTimeVariable()` and `QiStateVariable()`**
  - A variable that can be used during the control flow or to temporarily store a measured qubit state.
- **`QiPulse(length, shape, amplitude, phase)`**
  - A pulse object that can be used in other commands such as **Play**.

Basics (Context Managers)

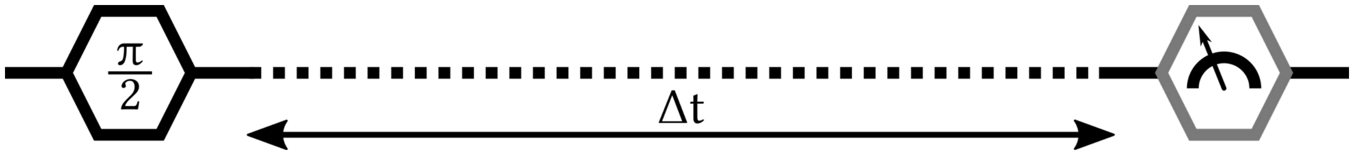
- **`with If(condition):`**
  - Conditional branching, only executes the indented block that follows if the condition is true.
- **`with Else():`**
  - Can follow after with If and does exactly what you would expect it to do.
- **`with ForRange(variable, start, end, step):`**
  - The passed variable will be looped from start to end (exclusive) with given increment (default: 1). The following block will be repeated for every value of the variable.

Rabi Experiment

```python
with QiJob() as rabi:
    q = QiCells(1)
    length = QiTimeVariable()
    with ForRange(length, 0, 100e-9, 4e-9):
        Play(q[0], QiPulse(length, frequency=q[0]["manip_frequency"]))
        Readout(q[0], save_to="result")
        Thermalize(q[0])
```

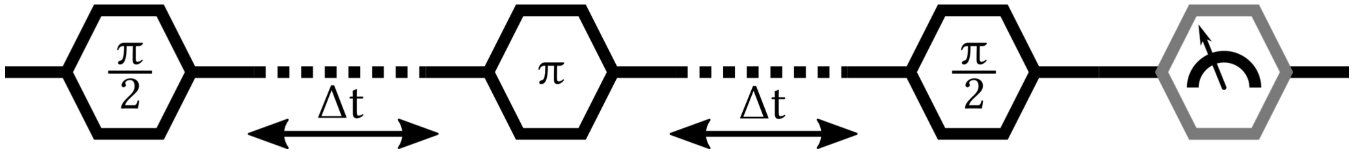
T1 Experiment

```
with QiJob() as job:
    q = QiCells(1)
    length = QiTimeVariable()
    with ForRange(length, start, stop, step):
        PiPulse(q[0])
        Wait(q[0], delay=length)
        Readout(q[0], save_to="result")
        Thermalize(q[0])
```



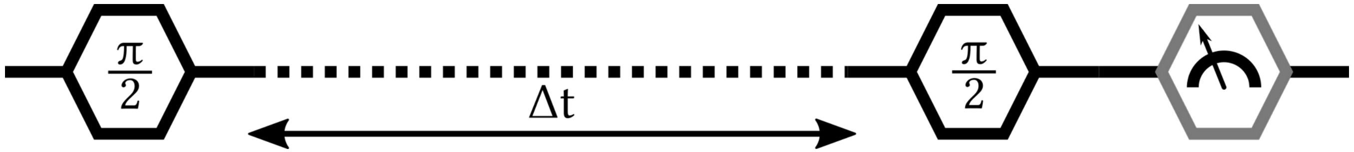
Spin Echo Experiment

```
with QiJob() as spin_echo:
    q = QiCells(1)
    length = QiTimeVariable()
    length_half = QiTimeVariable()
    with ForRange(length, start, stop, step):
        Assign(dst=length_half, calc=length >> 1)
        PiHalfPulse(q[0])
        Wait(q[0], delay=length_half)
        PiPulse(q[0])
        Wait(q[0], delay=length_half)
        PiHalfPulse(q[0])
        Readout(q[0], save_to="result")
        Thermalize(q[0])
```



Ramsey Experiment

```
with QiJob() as ramsey:
    q = QiCells(1)
    length = QiTimeVariable()
    with ForRange(length, start, stop, step):
        PiHalfPulse(q[0], detuning=detuning)
        Wait(q[0], delay=length)
        PiHalfPulse(q[0], detuning=detuning)
        Readout(q[0], save_to="result")
        Thermalize(q[0])
```

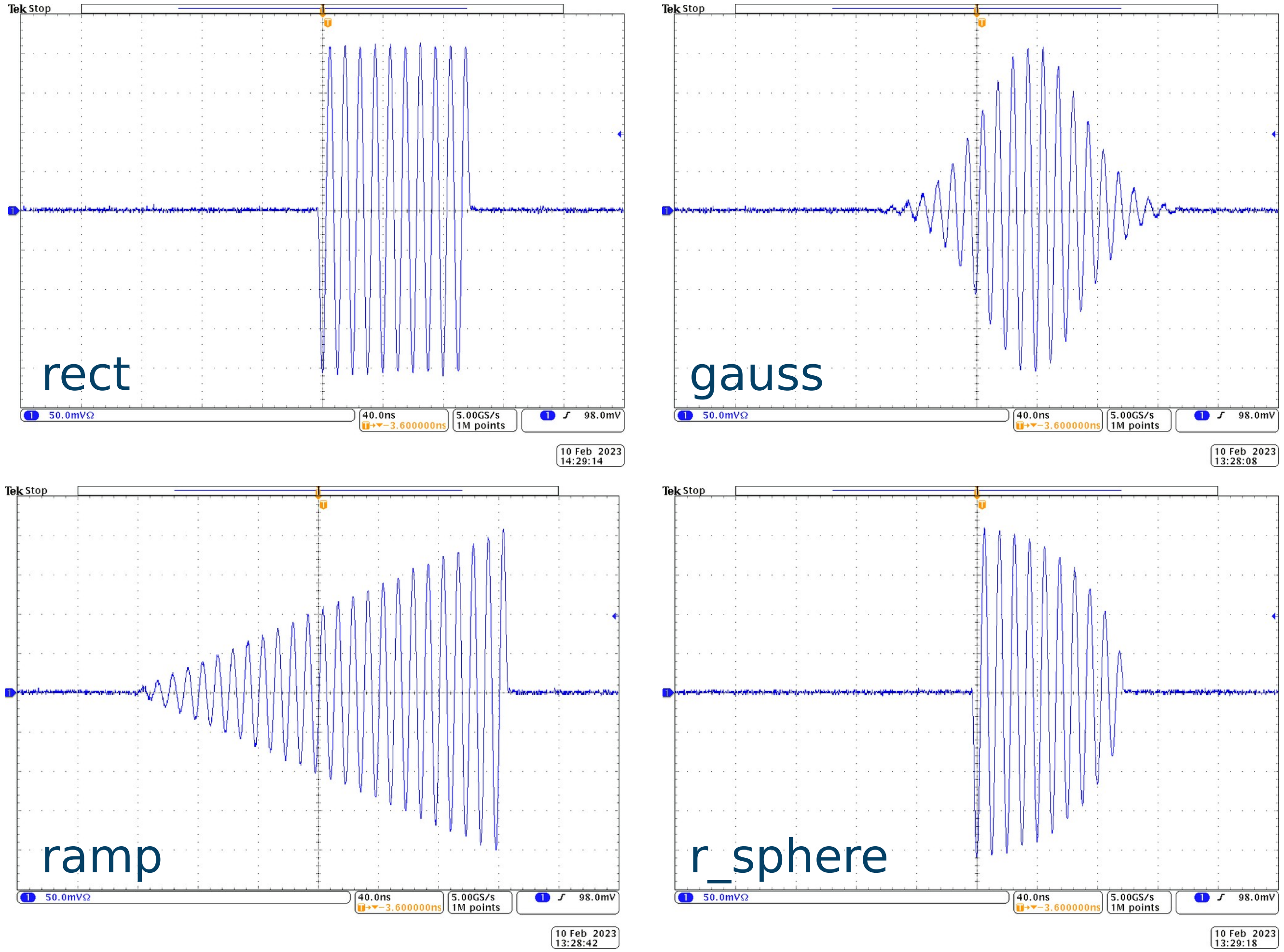


Pulse Shapes

With shorter pulses, Fourier components become a concern
→ alternative pulse shapes

- Envelopes are used to shape pulses
- 9 predefined envelopes
- User can create additional envelopes
- Maximal length currently ~4µs

```
Play(q[0], QiPulse(length=200e-9,
                   shape=ShapeLib.gauss,
                   frequency=100e6))
```



Current Application of QiCode: (Single) Qubit Characterisation

Preparational steps:

1. Describe the sample (**QiSample**)
2. Connect to the QiController (Ethernet)
3. Calibrate the amplitude of manipulation and readout pulses
4. Calibrate the electrical delay
5. Connect to and configure the analog RF frontend (Ethernet)

Characterisation:

1. Find resonator frequency f_{res}
2. Find qubit frequency f_{01}
3. Rabi
4. T1
5. SpinEcho
6. Ramsey

All steps for a setup are usually collected in a Jupyter notebook

Pulse Shapes

1. Describe the sample (**QiSample**)

```
#specify a 1D Sample object (1 Qubit chip)
sample = QiSample(1)

#necessary paramters to be provided
sample[0]["rec_pulse"] = 416e-9 #pulse length of the readout pulse
sample[0]["rec_length"] = 400e-9 #length of the recording window
sample[0]["T1"] = 20e-6 # Start with some conservative value so qubit can definitely thermalize
sample[0]["rec_frequency"] = 80e6
sample[0]["manip_frequency"] = 200e6
sample[0]["f_res"] = 8.5758e9
sample[0]["f_01"] = 6344.268e6

#calculate target LO frequencies
sample[0]["f_LO (R)"] = sample[0]["f_res"] + sample[0]["rec_frequency"]
sample[0]["f_LO (M)"] = sample[0]["f_01"] + sample[0]["manip_frequency"]

#you can also put in your own fields in this dict, as for example a subsample name
sample[0]["subsample"] = "test"
```

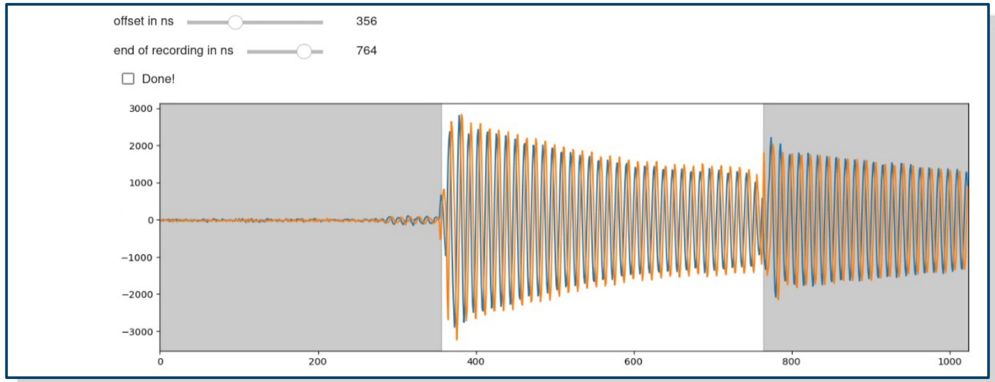
Preparational Steps

2.Connect to the QiController (Ethernet)

```
qic = ql.QiController('controller ip')
```

3.Calibrate the amplitude of manipulation and readout pulses for every unit cell

```
for cell in qic.cell:
    cell.readout.amplitude_calibration = (1,1) #(I,Q)
    cell.manipulation.amplitude_calibration = (1,1) #(I,Q)
```



4.Calibrate the electrical delay

```
#crop recording window to electrical delay
ql.init.crop_recording_window(qic, sample, averages=10000)

#calibration of global phase offset
ql.init.calibrate_readout_phase(qic, sample, averages=1000, set_sample=True)
```

Preparational Steps

5. Connect to and configure the analog RF frontend (Ethernet)

```
#Load board with correct driver
hf_manip = qkit.instruments.create(name="HF-PCB Manipulation", instype="RFPCB", address="10.22.197.146:4242")
hf_readout = qkit.instruments.create(name="HF-PCB Readout", instype="RFPCB", address="10.22.197.147:4242")

#configure readout board Demodulation path
hf_readout.set_demod_gain(7)
hf_readout.set_demod_attenuation(0)

#output attenuation
hf_readout.set_hf_attenuation(25)

#Set LO frequncies and power on
hf_readout.set_frequency(sample[0]["f_LO (R)"], offset=0)
hf_readout.on()

#manipulation board output attenuation
hf_manip.set_hf_attenuation(8)

#set LOs and power on
hf_manip.set_frequency(sample[0]["f_LO (M)"], offset=0)
hf_manip.on()
```

The Qkit framework is used to control the RF frontend (https://github.com/qkitgroup/qkit)

Characterisation

Karlsruhe Institute of Technology

1. Find resonator frequency f_{res}

```
#specification of scan parameters
cid = 0 #sample id
freq_center = sample[cid]["rec_frequency"]
freq_span = 20e6
freq_step = 0.1e6
averages = 1000

#Job specification to run on the platform
with QiJob() as job:
    q = QiCells(1)
    f = QiVariable()
    with ForRange(f, freq_center-freq_span/2, freq_center+freq_span/2, freq_step):
        PlayReadout(q[0], QiPulse(length=q[0]["rec_pulse"], frequency=f))
        Recording(q[0], duration=q[0]["rec_length"], offset=q[0]["rec_offset"], save_to="result")
        Wait(q[0], delay=10e-6)

exp = job.create_experiment(qic, sample, averages=averages, cell_map=[cid])

#The Qkit framework is used to execute the experiment (github.com/qkitgroup/qkit)
m = Measure_td(exp.qkit_sample)
m.measure_1D_AWG()
```

Characterisation

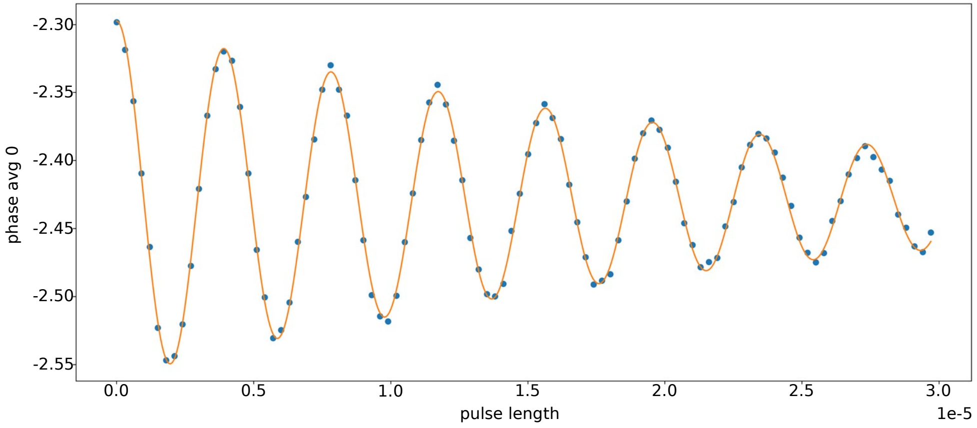
Karlsruhe Institute of Technology

3.Rabi

Drive Rabi oscillation to get the pi-pulse length for the sample.

1.

```
# sepcify parameters
t_max = 2e-6
t_step = 20e-9
averages = 5000
iterations = 1
cid = 0
qubit = sample[cid]["subsample"]
```



2.

```
def measure_rabi(t_max,t_step,averages,iterations,cid,qubit):
    exp = ql.jobs.Rabi(start=0, stop=t_max, step=t_step).create_experiment(
            qic, sample, cell_map=[cid], averages=averages)

    #The Qkit framework is used to execute the experiment
    m = Measure_td(qic.sample, exp.readout)
    m.measure_1D_AWG(iterations=iterations)

measure_rabi(t_max,t_step,averages,iterations,cid,qubit)

# Fit the oscillation with qfit, to extract the pi pulse length
qf = qfit.QFIT()
qf.load(entries=['pulse_length', f'phase_avg_0'])

qf.fit_damped_sine()
t_pi = 0.5/qf.popt[0]
print(f"Cell {cid}: t_pi = {t_pi*1e9:.2f} ns")

# if fit is fine the pi pulse length is saved in the sample object.
if input("Save pi pulse time in sample as pi? y/n") == "y":
    print(f"Ok, saving in cell {cid}!")
    sample[cid]["pi"] = t_pi
```

Conclusion

- QiCode is a Python based experiment description language developed to improve the usability of the QiController Hardware
- The modular approach of QiCode allows to create reusable building blocks (**QiGate**)
- Describing a qubit chip as a **QiSample** allows to simply reuse this description in various applications
- Some standard experiments (e.g. Rabi, T1, SpinEcho, Ramsey) are predefined in QiCode
- The language is already in use, mainly for the characterization of individual qubits

BACKUP

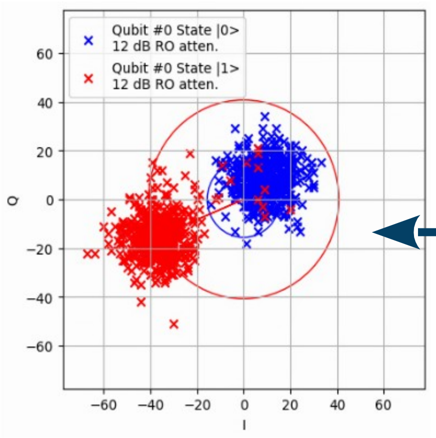
The Function ”calibrate_readout_phase“

- The readout phase is usually at some arbitrary phase, depending on the electrical delay
- This causes inconsistencies in the recorded data
- To avoid this, it is possible to normalize the phase of the read back signal

```
def calibrate_readout_phase(  qic: QiController,
                              sample: QiSample,
                              averages: int,
                              set_sample: bool = False,
                              cell: int = 0):
    qic_cell = sample[cell](qic)
    if qic_cell.recording.interferometer_mode:
        raise SystemError("Resetting the phase is not possible in interferometer mode.")
    with QiJob() as job:
        q = QiCells(1)
        Readout(q[0], save_to="result")
        Wait(q[0], 2e-6)
    job.run(qic, sample, averages, cell_map=[cell], data_collection="amp_pha")
    _, [pha_old] = job.cells[0].data("result")
    pha_old_calib = qic_cell.recording.phase_offset
    pha_calib = pha_old_calib - pha_old
    qic_cell.recording.phase_offset = pha_calib + 2 * np.pi
    if set_sample:
        sample[cell]["rec_phase"] = pha_calib
    job.run(qic, sample, averages, cell_map=[cell], data_collection="amp_pha")
    _, [pha_new] = job.cells[0].data("result")
    print(f"Phase was {pha_old:.5f} and is now calibrated to {pha_new:.5f}.")
    return pha_calib
```

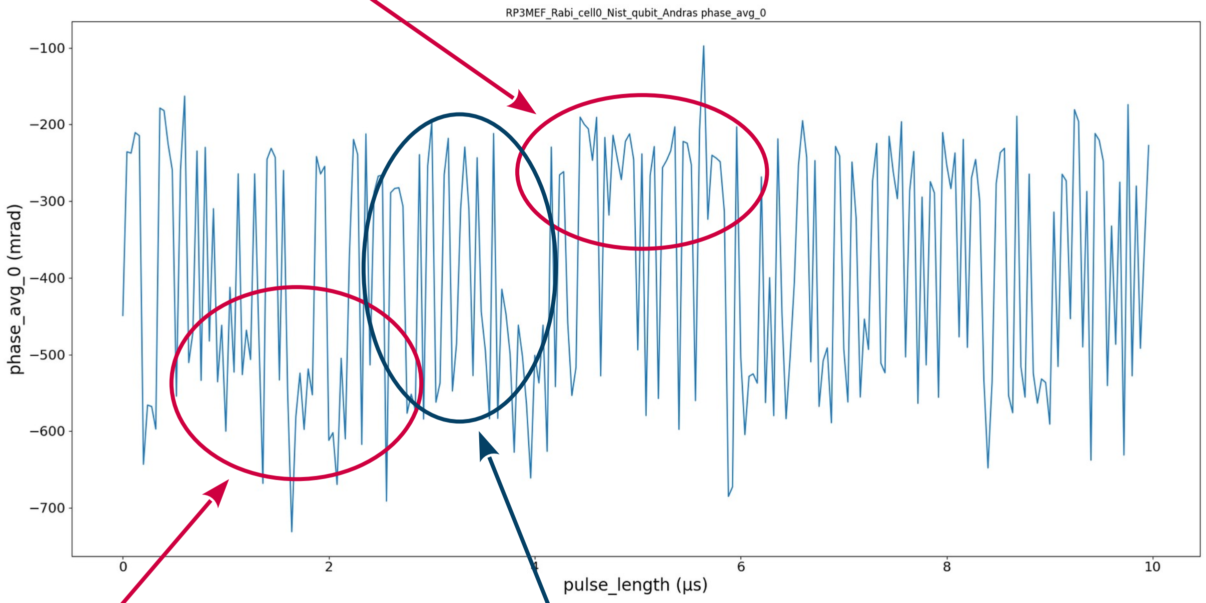
Singe Shot Rabi Measurement

- With the correct setup, single shot measurements are possible
- Mainly depending on amplification and dampening in the cryostat




```
ql.jobs.Readout(q[0], "result")
ql.jobs.Thermalize(q[0])
ql.jobs.PiPulse(q[0])
ql.jobs.Readout(q[0], "result")
ql.jobs.Thermalize(q[0])
```

Accumulation of one state



Accumulation of an other state

Equal distribution of both states