Introduction to PycQED

Adriaan Rol and Niels Bultink

March 2020

QUTech

QBLOX

PycQED is a python platform to control QC experiments

What is PycQED?

- Open source (MIT) Python platform for quantum computing experiments
- Build on top of QCoDeS
- Integration of OpenQL compiler

- Contains all basic functionality (instrument control, waveform management, data storage, live visualization, library of standard experiments + analysis)

- Widely used and tested
  - Transmons & spin qubits
  - TU Delft, ETH Zurich, UTSydney,

QCoDeS: https://qcodes.github.io/Qcodes/

OpenQL: https://github.com/QE-Lab/OpenQL

Providing the tools to control QC experiments

Design goals of PycQED

1. Extensible
   - Code is modular and reusable
   - Open source (MIT license)
2. Easy to use
   - Minimal number of concepts
   - Big library of standard experiments
3. Automatable
   - Closed loop between experiment and analysis (single language)
   - Minimal barrier between user and programmer (minimal user interface)

A minimal set of concepts

Core concepts

1. Parameter
2. Instrument
3. Measurement Control
4. Data storage & Analysis

An instrument is a container for parameters

Instrument & parameter

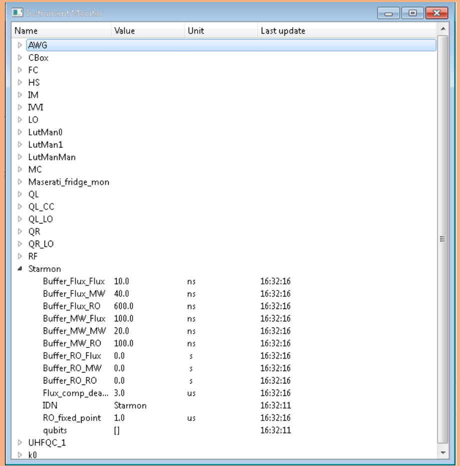
Parameter
- Represents a state variable of the system
- Can be gettable and/or settable
- Contains metadata such as units and labels

Instrument
- Container for parameters
- Provides standardized interface
- Provides logging of parameters (snapshot)
- Can correspond to physical hardware but can be more general

Example: Rhode & Schwarz Microwave source

```
from qcodes import VisaInstrument, validators as vals


class RohdeSchwarz_SGS100A(VisaInstrument):
    '''
    This is the qcodes driver for the
    '''
    def __init__(self, name, address,
        super().__init__(name, address

        self.add_parameter(name='frequ
                           label='Freq
                           units='Hz',
                           get_cmd='SO
                           set_cmd='SO
                           get_parser=
                           vals=vals.N
        self.add_function('reset', cal
        self.add_function('run_self_te

        self.connect_message()

    def parse_on_off(self, stat):
        if stat.startswith('0'):
            stat = 'Off'
        elif stat.startswith('1'):
            stat = 'On'
        return stat

    def set_status(self, stat):
        if stat.upper() in ('ON', 'OFF
            self.write(':OUTP:STAT %s'
        else:
            raise ValueError('Unable t
                             'expected "ON" or "OFF"')
```

PycQED provides an instrument monitor



Note: Instruments and parameters are implemented using QCoDeS classes. However, the notion of Instruments is generalized in PycQED.

All experiments share a similar structure

Measurement Control

Every experiment consists of:

1. Some parameter(s) is/are varied
2. Some parameter(s) is/are measured
3. Data is saved and analysed

Example: Heterodyne spectroscopy experiment

PycQED syntax

```
MC.set_sweep_function(source.frequency)
MC.set_sweep_points(np.linspace(start, stop, steps))
MC.set_detector_function(HeterodyneDetector())
MC.run(name='Heterodyne')
ma.MeasurementAnalysis()
```

Measurement Control

- Enforces structure
- Standardizes data storage
- Provides life plotting
- Supports "advanced" experiments
  - Software controlled
  - Hardware controlled
  - 1D/2D/nD
  - Adaptive loop

Demo 1: The Measurement Control

Branch: develop ▾ PycQED_py3 / examples / MeasurementControl.ipynb Find file Copy path

caenrigen NEW MC adaptive tutorial and examples cleanup e49bd23 4 days ago

1 contributor

432 lines (432 sloc) | 12 KB Raw Blame History

Tutorial 1. The Measurement Control

This tutorial covers basic usage of PycQED focusing on running basic experiments using `MeasurementControl`. The `MeasurementControl` is the main `Instrument` in charge of running any experiment. It takes care of saving the data in a standardized format as well as live plotting of the data during the experiment. `PycQED` makes a distinction between `soft`(ware) controlled measurements and `hard`(ware) controlled measurements.

In a `soft` measurement `MeasurementControl` is in charge of the measurement loop and consecutively sets and gets datapoints. A `soft` measurement can be 1D, 2D or higher dimensional and also supports adaptive measurements in which the datapoints are determined during the measurement loop.

Experiments are stored together with metadata

Data storage

- Every experiment has it's own folder
- Timestamp is used as a unique ID
- Data files contain
  - Dataset (arrays and metadata)
  - Snapshot of all instruments
  - Analysis results (optional)
- Dataformat implemented in HDF5
  - Easy read and write functions
  - Helpers to find files and extract specific parameters

Example: Folder structure of data directory

Name	Date Modified	Size
test_data	20 Jan 2020 at 13:43	
20161124	17 Nov 2019 at 17:29	
20161214	17 Nov 2019 at 17:29	
20170120	17 Nov 2019 at 17:29	
20170201	17 Nov 2019 at 17:29	
20170227	Today at 22:29	
20170328	17 Nov 2019 at 17:29	
20170412	17 Nov 2019 at 17:29	
20170413	17 Nov 2019 at 17:29	
20170501	17 Nov 2019 at 17:29	
20170606	20 Jan 2020 at 13:43	
162250_SSRO_QL_QR	17 Nov 2019 at 17:29	
20170607	Today at 22:29	
145645_Ramsey_10pulse_sep_QR	17 Nov 2019 at 17:29	
145645_Ramsey_10pulse_sep_QR.hdf5	17 Nov 2019 at 17:29	290
152324_T1_QL	17 Nov 2019 at 17:29	
152943_echo_QL	17 Nov 2019 at 17:29	
160504_Rabi-n1_QR	17 Nov 2019 at 17:29	
161234_Motzoi_XY_QR	17 Nov 2019 at 17:29	
161456_AllXY_QR	17 Nov 2019 at 17:29	
210448_T1_QR	17 Nov 2019 at 17:29	
210555_Motzoi_XY_QR	17 Nov 2019 at 17:29	
210655_RB_100seeds_QR	17 Nov 2019 at 17:29	

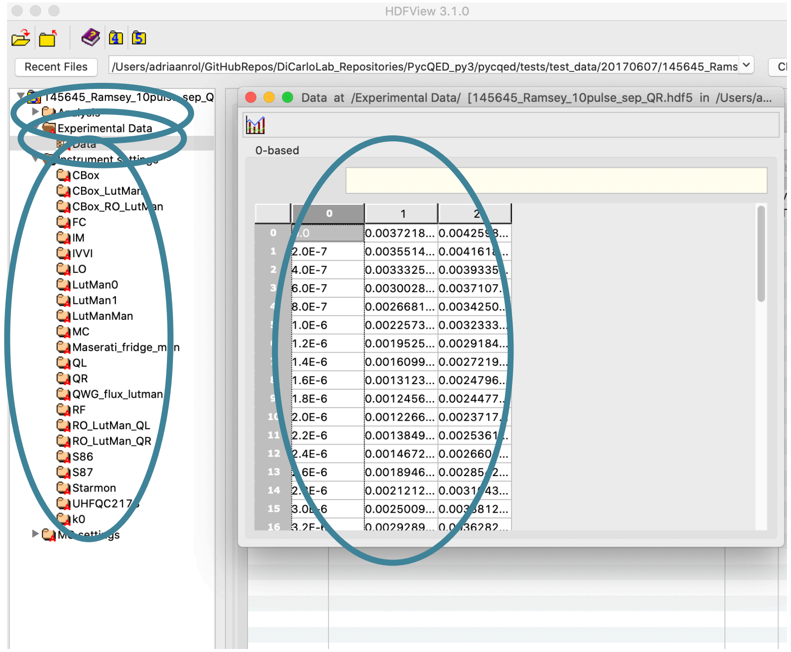
https://www.hdfgroup.org/

Experiments are stored together with metadata

Data storage

- Every experiment has it's own folder
- Timestamp is used as a unique ID
- Data files contain
    - Dataset (arrays and metadata)
    - Snapshot of all instruments
    - Analysis results (optional)
- Dataformat implemented in HDF5
    - Easy read and write functions
    - Helpers to find files and extract specific parameters

Example: Structure of individual data file



https://www.hdfgroup.org/

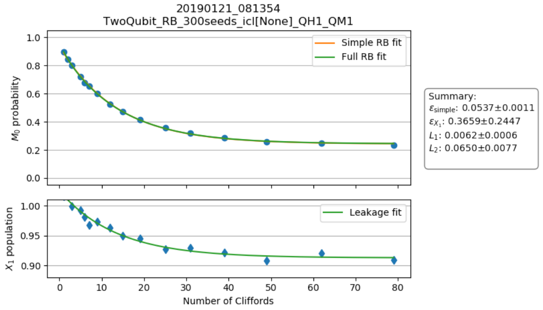
QuTech QBLOX

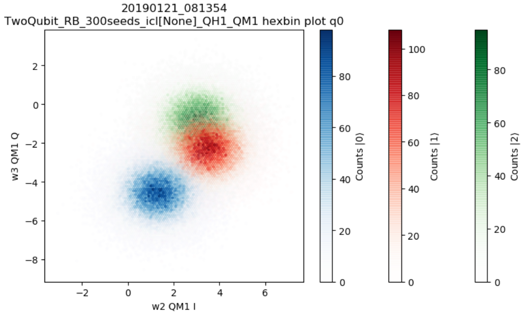
PycQED contains a large library of standard analyses

Analysis

- Share common structure
- Helpers for finding and extracting data
- Fitting using the lmfit library
- Standard figures for each analysis

Example: Analysis for two-qubit randomized benchmarking with leakage modification



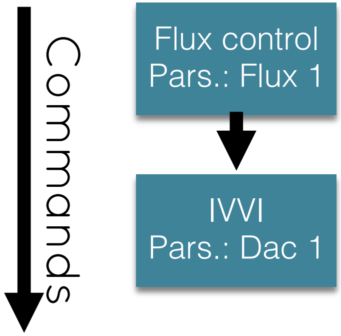


Meta instruments allow layers of abstraction and modularity

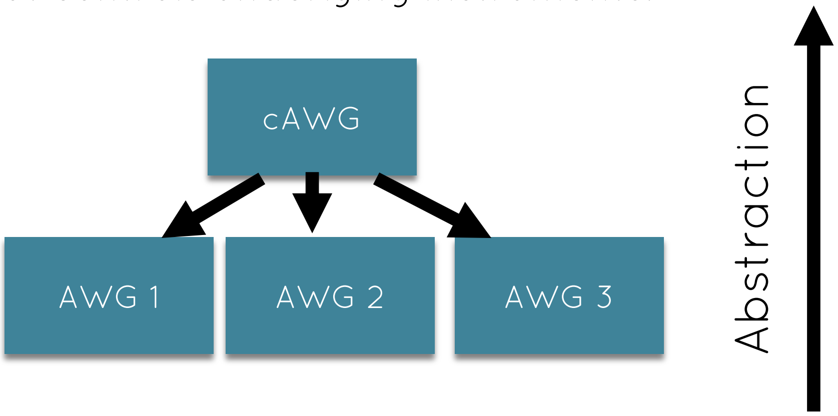
Meta instruments

A **meta-instrument** can contain other instruments but acts like a regular instrument

Example 1: Flux control
Converts flux to dac-voltages using a calibrated correction matrix



Example 2: Composite AWG
Acts as a single multi-channel AWG but controls underlying instruments.



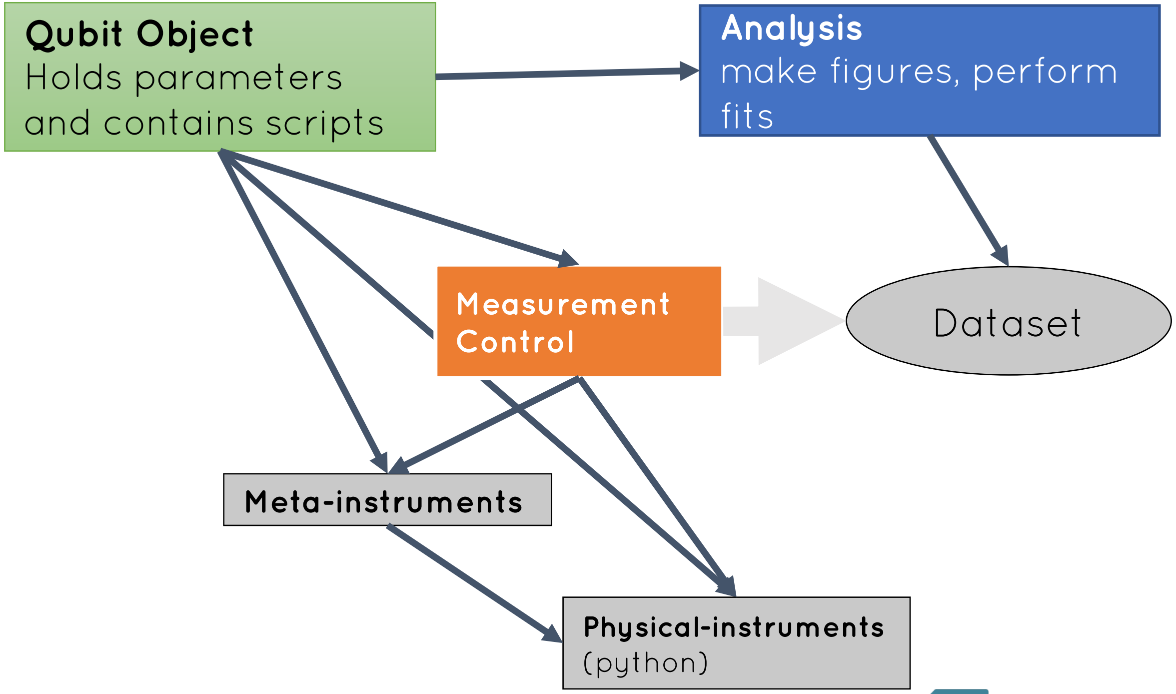
QUTech
QBLOX

The Qubit object is an instrument that contains experiments

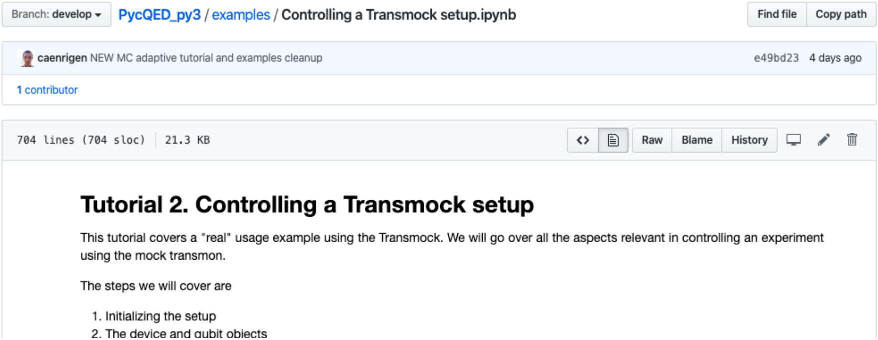
Qubit object

Typical script

1. Prepare by setting parameters
2. Execute Loop
3. Analyze dataset



Demo 2: Controlling a Transmock setup

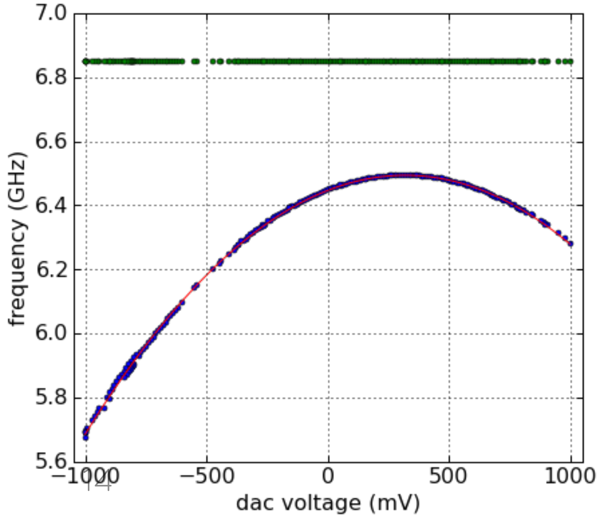


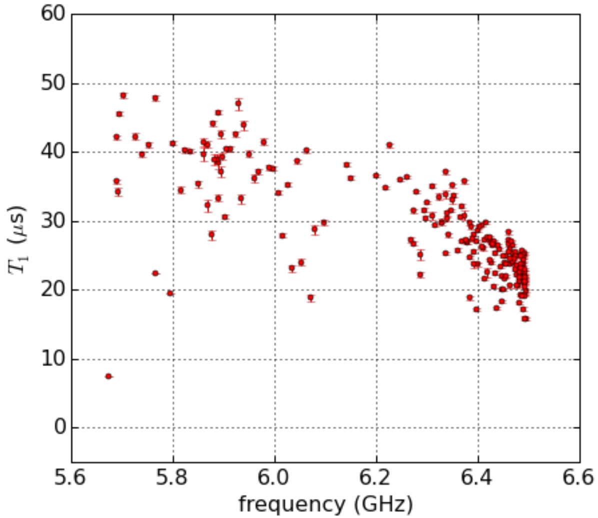
Nested loops using a python for loop

Standard loop:

T1/T2/T2* as a function of flux

```
currents=np.linspace(0.0021,0.003,10)
for current in currents:
    fluxcurrent.FBL_QM1(current)
    qubit.msmt_suffix='current {}'.format(current)
    qubit.find_frequency(freqs=np.arange(5.55e9,5.69e9,1e6))
    qubit.measure_T1(times=np.arange(0,60e-6,1.0e-6))
    qubit.measure_ramsey(times=np.arange(0.02e-6,5e-6,0.06e-6))
    qubit.measure_echo(times=np.arange(0.04e-6,30e-6,0.48e-6))
```

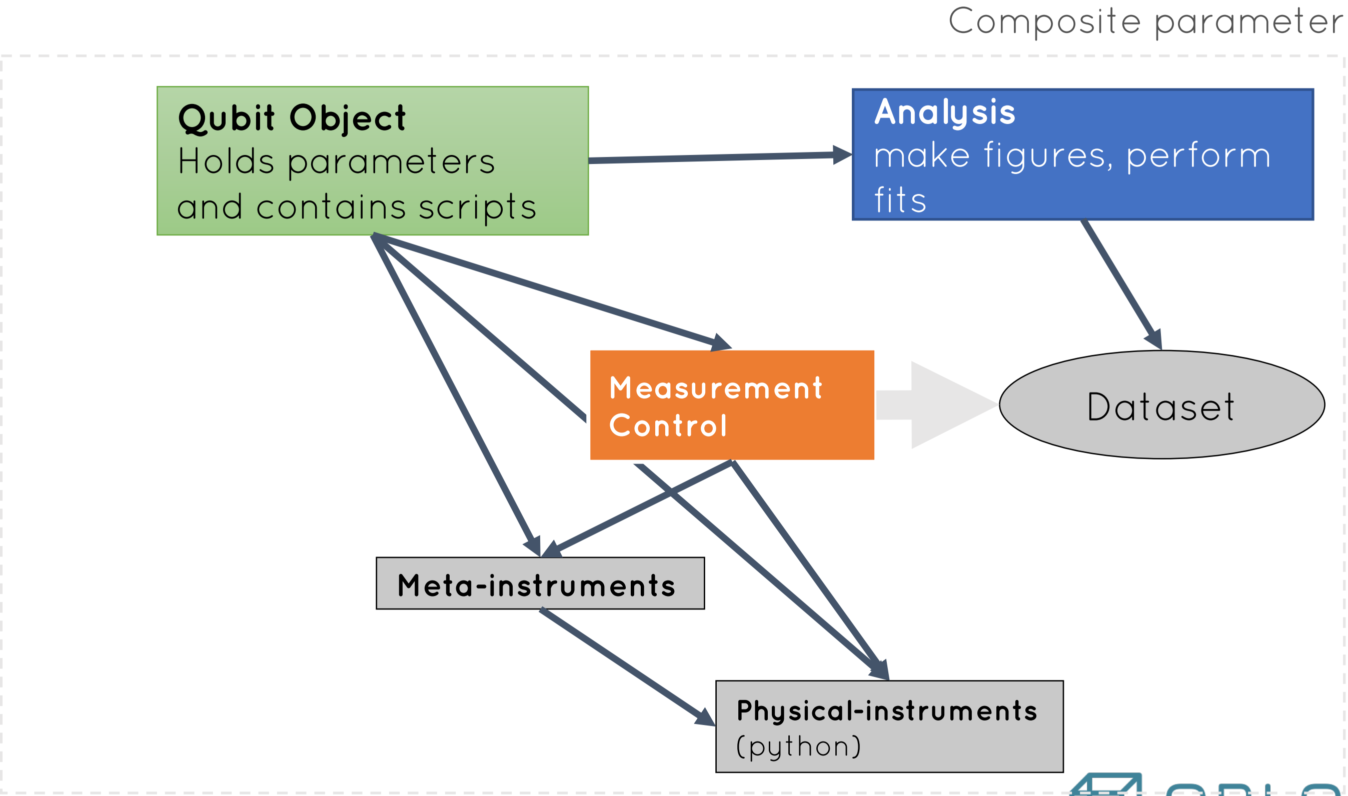




The Qubit object is an instrument containing experiments

Qubit object

Typical script
1. Prepare by setting parameters
2. Execute Loop
3. Analyze dataset

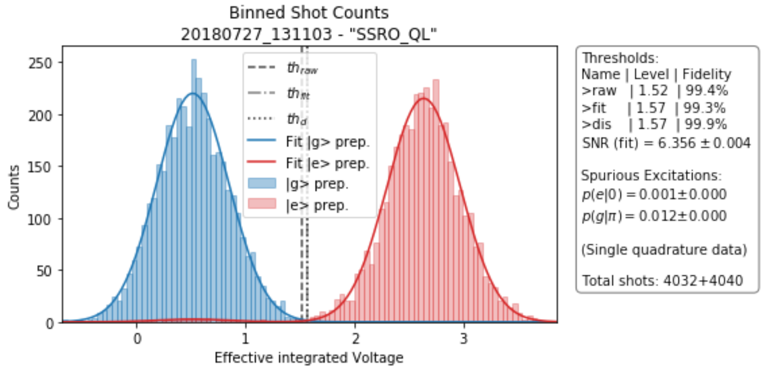


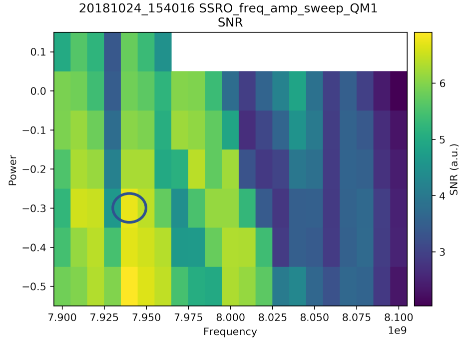
Nested loops using PycQED

Function detector

Perform Single-Shot readout as a function of TWPA bias

```python
d = det.Function_Detector(
    self.measure_ssro,
    msmt_kw={
        'nr_shots': nr_shots,
        'analyze': True, 'SNR_detector': True,
        'cal_residual_excitation': True,
        'prepare': False,
        'disable_metadata': True
    },
    result_keys=['SNR', 'F_d', 'F_a']
)
nested_MC.set_sweep_function(pump_source.frequency)
nested_MC.set_sweep_points(freqs)
nested_MC.set_detector_function(d)
nested_MC.set_sweep_function_2D(pump_source.power)
nested_MC.set_sweep_points_2D(powers)
label = 'SSRO_freq_amp_sweep' + self.msmt_suffix
nested_MC.run(label, mode='2D')
```



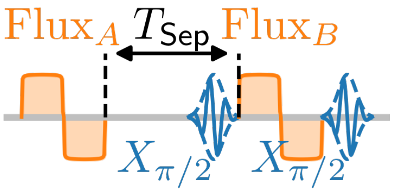


QUTech

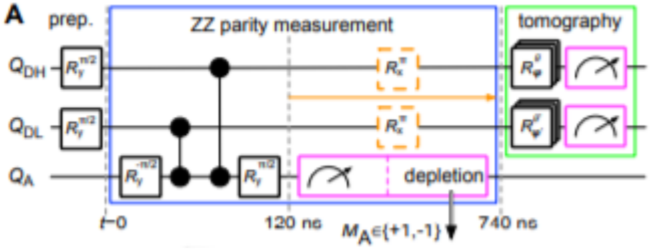
QBLOX

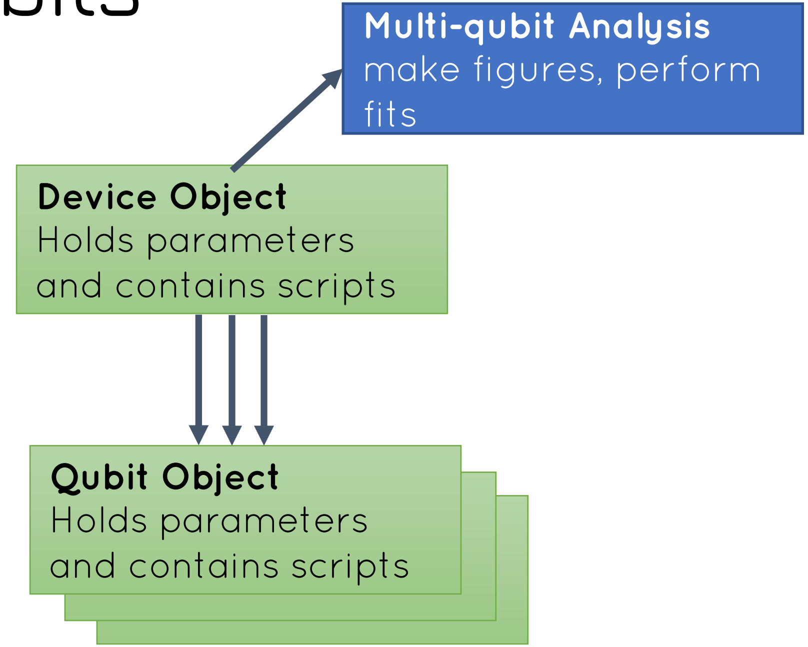
Device object –multiple qubits

- 2Q gate tuning



- 2Q Randomized Benchmarking
- 3Q Parity measurements





Calibrations can be automated using autodepgraph

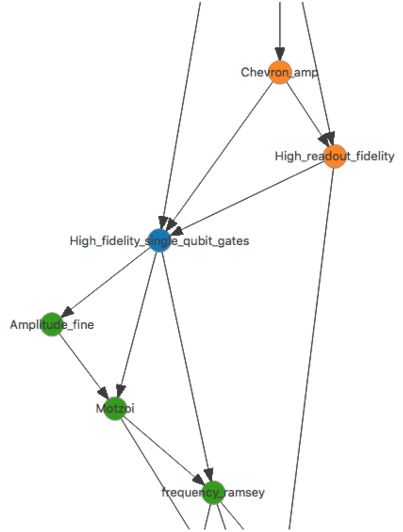
Automated calibration

AutoDepGraph build passing codacy A coverage 92%

https://github.com/AdriaanRol/AutoDepGraph

Features

- Node logic
- Framework + loading and storing of graphs
- Live monitor of graph
- Built for use with PycQED



Autodepgraph: https://github.com/AdriaanRol/AutoDepGraph

Inspired by Kelly et al. ArXiv 1803.03226

Sequencing

CC-based

Qubit object

```
def measure_T1(self, times=None, MC=None,
               analyze=True, close_fig=True, update=True,
               prepare_for_timedomain=True):
    # docstring from parent class
    # N.B. this is a good example for a generic timedomain
    # the CCL transmon.
    if MC is None:
        MC = self.instr_MC.get_instr()
```

OpenQL API (python)

```
p = oqh.create_program('T1', platf_cfg)

for i, time in enumerate(times[:-4]):
    k = oqh.create_kernel('T1_{}'.format(i), p)
    k.prepz(qubit_idx)
    wait_nanoseconds = int(round(time/1e-9))
    k.gate('rx180', [qubit_idx])
    k.gate("wait", [qubit_idx], wait_nanoseconds)
    k.measure(qubit_idx)
    p.add_kernel(k)
```

eQASM

```
k_T1_0:
    1     prepz s0
    qwait 9999
    1     cw_01 s0
    1     measz s0
    qwait 15

k_T1_1:
    1     prepz s0
    qwait 9999
    1     cw_01 s0
    2     measz s0
    qwait 15
```

Qubit Object
Holds parameters and contains scripts

Compiler
Creates eQASM file

Physical-instrument:
Central Controller

QUTech

QBLOX

Waveform management

Lookup table managers

```python
default_mw_lutmap = {
    0  : {"name" : "I"      , "theta" : 0        , "phi" : 0 , "type" : "ge"},
    1  : {"name" : "rX180"  , "theta" : 180      , "phi" : 0 , "type" : "ge"},
    2  : {"name" : "rY180"  , "theta" : 180      , "phi" : 90, "type" : "ge"},
    3  : {"name" : "rX90"   , "theta" : 90       , "phi" : 0 , "type" : "ge"},
    4  : {"name" : "rY90"   , "theta" : 90       , "phi" : 90, "type" : "ge"},
    5  : {"name" : "rXm90"  , "theta" : -90      , "phi" : 0 , "type" : "ge"},
    6  : {"name" : "rYm90"  , "theta" : -90      , "phi" : 90, "type" : "ge"},
    7  : {"name" : "rPhi90", "theta" : 90       , "phi" : 0 , "type" : "ge"},
    8  : {"name" : "spec"   , "type"  : "spec"}  ,
    9  : {"name" : "rX12"   , "theta" : 180      , "phi" : 0 , "type" : "ef"},
    10 : {"name" : "square", "type"  : "square"},

}
```

```python
def _add_waveform_parameters(self):
    # defined here so that the VSM based LutMan can overwrite this
    self.wf_func = wf.mod_gauss
    self.spec_func = wf.block_pulse

    self._add_channel_params()
    self.add_parameter('cfg_sideband_mode',
                       vals=vals.Enum('real-time', 'static'),
                       initial_value='static',
                       parameter_class=ManualParameter)
    self.add_parameter('mw_amp180', unit='frac', vals=vals.Numbers(-1, 1),
                       parameter_class=ManualParameter,
                       initial_value=0.1)
    self.add_parameter('mw_amp90_scale',
                       vals=vals.Numbers(-1, 1),
                       parameter_class=ManualParameter,
                       initial_value=0.5)
    self.add_parameter('mw_motzoi', vals=vals.Numbers(-2, 2),
                       parameter_class=ManualParameter,
                       initial_value=0.0)
    self.add_parameter('mw_gauss_width',
                       vals=vals.Numbers(min_value=1e-9), unit='s',
                       parameter_class=ManualParameter,
                       initial_value=4e-9)
```

Qubit Object
Holds parameters and contains scripts

Lookup table manager
Abstraction of one or more physical instruments

Physical-instruments
QWG/Pulsar/HDAWG

Pro's/cons

- ✓ Minimal user interface
- ✓ Open source (MIT license)
- ✓ Large libraries of:
    - Experiments
    - Analyses
    - Instrument drivers
- ✓ operable from command line or Jupyter Notebook

- ✗ Minimal user interface
- ✗ Contains dead code
- ✗ No centralized maintenance
- ✗ Limited documentation