User Manual

Zhaohui Yang, [zhy@email.arizona.edu](mailto:zhy@email.arizona.edu)

[Quantum Information and Materials Group](https://quantum.lab.arizona.edu/), University of Arizona

*Last updated: May 2022*

This manual consists of the design framework, implementation details, usage examples, necessary configuration descriptions and API documentation of the GUI software for ODMR experiments.

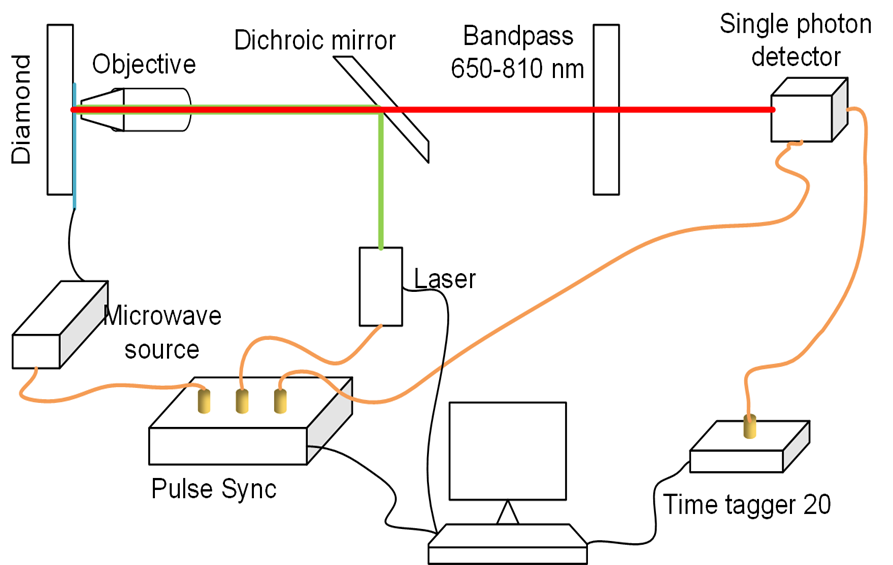
# Design Framework

The initial goal of this project is to provide an as universal as possible SDK of the series of usual scheduling algorithms for [ODMR](https://en.wikipedia.org/wiki/Optically_detected_magnetic_resonance) and spin manipulation experiments. Currently the SDK project (named [Odmactor](https://github.com/Youngcius/odmactor)) supports all the frequency-domain and time-domain ODMR operation functionalities and all of them have been applied in the [QIM](https://quantum.lab.arizona.edu/) group at UA. Furthermore, a desktop software with user-friendly GUI building on this SDK and Qt graphic library was also implemented and tested well in QIM lab. It is called [Odmactor-GUI](https://github.com/Youngcius/odmactor-gui) in GitHub. Following content includes descriptions of the algorithms design and GUI implementation as well as the integration approach.

## Hardware and software support

This project is designed for physical experiments, thus it does not depend on only software techniques. The following figure shows usual necessary devices of a ODMR platform:

1. Microwave source (MW): also called RF signal generator. The [SMB 100A](https://www.rohde-schwarz.com/us/products/test-and-measurement/analog-signal-generators/rs-smb100a-microwave-signal-generator_63493-9379.html) provide by [R&S](https://www.rohde-schwarz.com/) is what we used currently.
2. Arbitrary sequence generator (ASG): the most crucial device, used for generating control sequences. Here in we use the ASG8005 provided by [CIQTK](https://en.ciqtek.com/)
3. Avalanche photodiode (APD) + Time Tagger: the most usual photon counting scheme. We use a semiconductor APD and an [8-channel Time Tagger](https://www.swabianinstruments.com/time-tagger/) provided by [Swabian](https://www.swabianinstruments.com/).
4. Lock-in amplifier + DAQ: another data acquisition scheme. When signals are weak, the APD + Tagger cannot detect effective signals, we use this lock-in scheme for detecting. The lock-in amplifier we use is [RS830](https://www.thinksrs.com/products/sr810830.htm) provided by [Stanford Research](https://www.thinksrs.com). The DAQ is a regular device provided by [National Instruments](https://www.ni.com).
5. Laser: 532nm laser or 730nm laser. In the detecting on NV in diamond, we use the 532nm fiber pulse laser provided by CIQTK; in the detecting on SiV in SiC, we use the 730nm free-space solid laser. They are both modulated by ASG’s output sequences via connection to the laser itself or AOM.



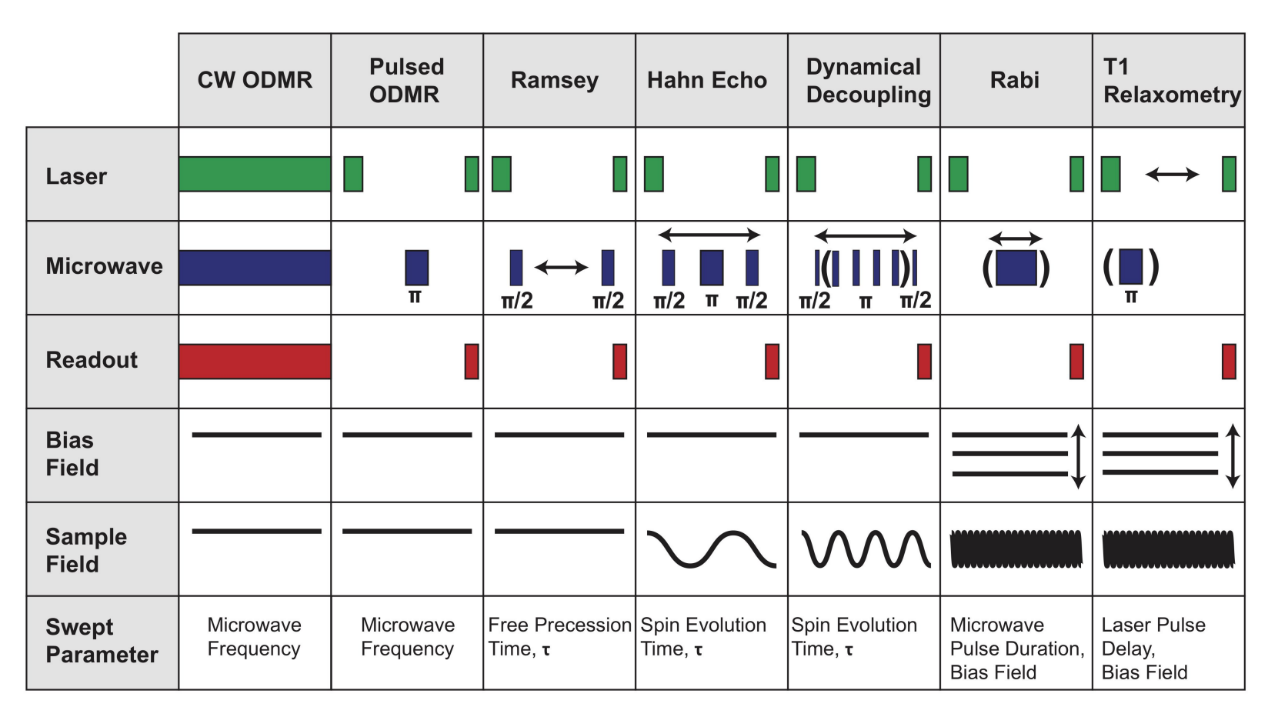
Except for the [Time Tagger](https://www.swabianinstruments.com/static/documentation/TimeTagger/index.html), all the corresponding instrument vendors do not provide user-friendly programming interfaces. To use them in an autonomous manner, we have to program from scratch, building on libraries related to soft-hardware interfaces, e.g., pymeasure, pyvisa, RsInstrument. The core soft-hardware programming is on ASG, via a compiled module provided by CIQTK.

Other necessary libraries are those standard and usual third-party ones. All the dependencies are listed in file requirements.txt.

## Odmactor SDK design

### OOP implementation

This SDK is an OOP Python project. It mainly implements a set of schedulers by abstracting the universal pipelines and usual settings of relevant ODMR experiments. Usually, ODMR experiments are divided into “frequency-domain” and “time-domain” two detecting modes, as shown in the figure below. Both needs Laser/MW/ASG sequences settings and photon counts readout. In the former mode, the MW frequency needs to change to acquire a ODMR spectrum. In the latter mode, what needs to be changed is time, which can be MW operation time or spin evolution time or other delay times. Thus, in an OOP insight, they are implemented in a set of “scheduling” classes with well-defined inheritance relationships.



Technically, the time-domain measurements are not called ODMR experiments, while they are performed in the ODMR platform. Therefore, we unitedly call them ODMR measurements. Usages of them are briefly summarized as follows.

*Frequency-domain ODMR:*

1. Continuous-wave detecting:
   1. Polarize spin systems
   2. Operate continuous MW and readout signals for the whole sequence period
2. Pulse detecting:
   1. Polarize spin systems
   2. Apply a fixed-interval MW pulse
   3. Readout final fluorescence signals

*Time-domain ODMR:*

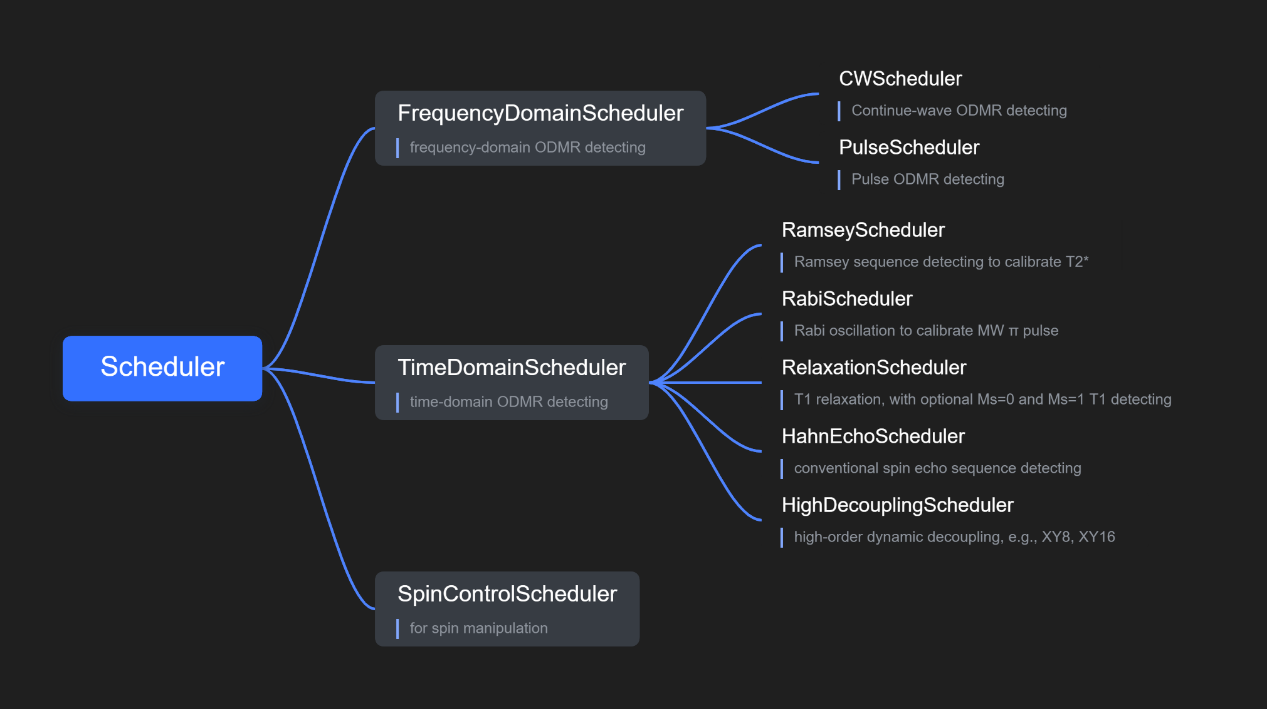
1. Rabi oscillation:
   1. Initialize spin systems to ground states
   2. Operate a MW pulse with variable interval
   3. Readout final spin states (population), i.e., fluorescence signals in ODMR
2. Ramey detecting:
   1. Initialize spin systems to ground states
   2. Initialize spin systems to equal-amplitude superposition state using the calibrated MW pulse
   3. Wait for a time interval
   4. Operate a calibrated MW pulse again
   5. Readout final spin states
3. T1 relaxation:
   1. Initialize spin states to ground states
   2. Initialize spin states to excited states using the calibrated MW pulse (optional)
   3. Wait a time interval
   4. Readout final spin states
4. Hahn echo:

Sequences are similar with those for Ramsey detecting, while there is an additional MW pulse between the two pulses.

1. High-order dynamical decoupling:

Sequences are similar with those for Ramsey detecting, while there are additional MW pulse between the two pulses.

### Scheduler classes



As the OOP modeling insight, the series of scheduler classes are implemented with such inheritance relationships. Classes Scheduler, FrequencyDomainScheduler, TimeDomainScheduler, and SpinControlScheduler are all abstract base classes, which cannot be directly used by users. Instead, user should use the application-level classes (CWScheduler, PulseScheduler, HahnEchoScheduler, etc.) to run ODMR scheduling pipelines. Notably, the abstract class SpinControlScheduler is reserved for fine spin manipulation. It might be required for future quantum information processing experiments building on single spin systems. Due to current restricted experimental conditions and requirements, it is not yet implemented.

A picture containing table

Description automatically generated

The following structure demonstrates part of the necessary data fields, general scheduling methods (methods of the base class) and specific scheduling methods. By means of the SDK, there is a set of standard scheduling pipelines, for which please refer the [example files](https://github.com/Youngcius/odmactor/tree/master/example) or the [README of Odmactor](https://github.com/Youngcius/odmactor/blob/master/README.md).

### Modules structure

This Python SDK is composed of three packages scheduler, instrument, utils. Package scheduler consists of corresponding modules which are the implementation of the series of scheduler classes above. Package scheduler is the software-level abstraction of instruments, including ASG, MW, Lock-in Amplifier, Time Tagger, etc. Although for some of them, instrument providers have provided user-friendly SDK or programming guidance, the usage is not convenient and more importantly, not consistent. Herein we construct abstract instrument classes with universal operation methods, i.e., the instruments can be operated in an OOP approach. For instance, all the abstract instrument classes in our project consist of the same-name member methods like start, stop and close. And each instrument is automatically connected once its corresponding software instance is initialized. Package utils consists of utils functions necessary in this SDK, particularly the functions for processing ASG sequences. In the GUI software, another package ui consists of predefined UI description files (“.ui” and “.py”), implemented drawing on [QtDesigner](https://doc.qt.io/qt-5/qtdesigner-manual.html) and [PyUIC](https://pypi.org/project/pyuic5-tool/).

├─instrument  
│ │ asg.py  
│ │ laser.py  
│ │ lockin.py  
│ │ microwave.py  
│  
├─scheduler  
│ │ base.py  
│ │ customization.py  
│ │ decouple.py  
│ │ frequency.py  
│ │ spin.py  
│ └─ time.py  
│  
└─utils  
 │ plotting.py  
 │ sequence.py  
 │ utils.py  
 │  
 ├─asg  
 │ ASG8005\_PythonSDK.py  
 │ ASGDLL\_x64.dll  
 └─ ASGDLL\_x86.dll

Each module and each function in it consists of detailed docstrings and even parameter type declarations. User can reach to the source code or generate vivid Sphinx-style API documentation.

## GUI overview

A screenshot of a computer

Description automatically generated

The main body of this UI consists of “configuration list” and “demonstration pages” two parts, which are implemented based on “Tool Box” and “Tab Widget” Qt components, respectively. In the “configuration list” part (blue framed), user can perform generic operations on instruments, configure ODMR detecting sequences and scanning range, and perform spin manipulation (not implemented yet). In the “demonstration pages” (green framed), use can have a look at real-time photon counting, ODMR detecting results, and sequences visualization. Details of them are described in the next section.

# Operation Guiding

## Odmactor-GUI functionalities

Once familiar with the entire ODMR operation mode, functionalities of corresponding components (widgets) are demonstrative. There are still some components worthy additional descriptions.

### Configuration list

*MW generic settings:*

pulse is not necessary to be set except for Ramsey/HahnEcho/HighDecoupling modes.

When “timer” is checked, MW will automatically stop after a set time interval once the push button “on/off” is checked.

Graphical user interface, application

Description automatically generated

Laser generic settings:

Graphical user interface

Description automatically generated

Current Lasers do not support direct programming control. But if they do, required control functionalities should be as the above components show. It is a reserved UI module.

*ASG channels settings:*

Graphical user interface

Description automatically generated

Our ASG has 8 output channels. Laser, MW, APD and Tagger are controlled by ASG in most cases. ASG can also generate lock-in synchronic sequences to MW and Lock-in Amplifier, but usually it is not economic. Instead, user often use an outside signal generator to generator lock-in synchronic signals. All control sequences can be set to high-effective or low-effective arbitrarily.

*Time Tagger input channels:*

Table

Description automatically generated with low confidence

Out Tagger has 8 input channels that can be fed by detected signals or trigger signals. The input channel settings are nontrivial only if the radio button “Use Tagger” is checked. Otherwise, the backend detecting is finished by Lock-in Amplifier + DAQ.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, table

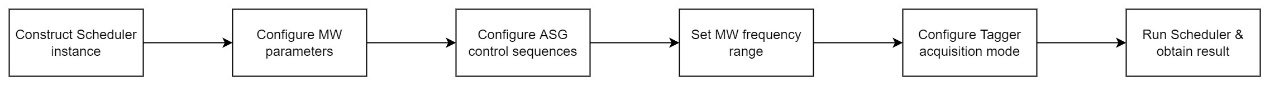
Description automatically generated

Sequence parameter setting examples are given in the next section.

### Demonstration pages

Which is implemented based on the “Tab Widget” Qt component.

## Frequency-domain ODMR



### Continuous-wave ODMR detecting

Chart, bar chart

Description automatically generated

Chart, line chart

Description automatically generated

### Pulse ODMR detecting

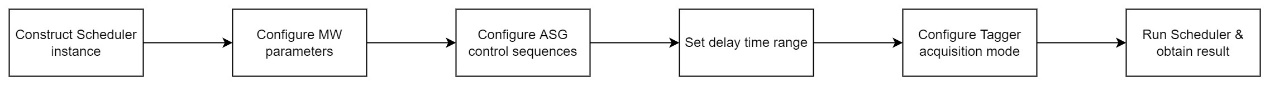
Chart

Description automatically generated

Chart, line chart

Description automatically generated

## Time-domain ODMR



### Rabi oscillation

Chart

Description automatically generated

Chart, line chart

Description automatically generated

### Ramsey detecting

Chart

Description automatically generated

### T1 relaxation

Ms=0 or Ms=1, MW on/off

Chart

Description automatically generated

### Hahn echo

Check the radio button “Hahn echo”. Other steps are the same as Ramsey detecting.

Chart

Description automatically generated

### High-order dynamical decoupling

Check the radio button “High-order DD”. Set desirable “decoupling order”. Other steps are the same as Ramsey detecting.

When the order is 1 (the default value), it is equivalent to Hahn echo measurement; when the order is 0, it is equivalent to Ramsey measurement. The following figure shows an instance of 8-order decoupling sequences.

Chart

Description automatically generated

## Other scheduling modes