FAIR-Battery

Sanli Faez

Mar 02, 2023

FURTHER IN THE DOCS:

In the *FAIR-Battery* FAIR-Battery project, we aim to create an open-source electrochemical battery (FAIR = Findable + Accessible + Interoperable + Reproducible).

CHAPTER

ONE

FAIR-BATTERY IS MEANT TO BE:

- Open-source: all designs and data that is useful for further development will be shared
- Scalable: both small and medium scale installations are supported and considered in new designs
- Versatile: alternative chemistries or material resources will be accommodated as much as possible

You can find the code of this package on Github. The documentation is hosted on Read The Docs.

1.1 Introduction to FAIR-Battery

In the FAIR-Battery project, we aim to create an open-source electrochemical battery (FAIR = Findable + Accessible + Interoperable + Reproducible). We seek to present an open-hardware platform for a versatile battery technology and make the platform radically accessible: 1- by deliberately using low cost and locally available materials suitable for local user groups, and 2- by setting up the education communities on top of the open-hardware design.

On this route, we thrive to not only provides the necessary technical details for engineering and production, but also incorporates the local constraints for actually adopting and using the technology. These constraints relate to language, availability of materials and expertise, maintenance capacity, or other locally varying conditions, which must be identified as part of the project. Our envisioned FAIR-Battery platform will track and seek to remove these constraints in each stage of the development by direct consultation with the user-groups.

1.1.1 FAIR-Battery strategic choices

In order to stay focused on the main goals of the FAIR-Battery project, and to not get distracted by feature-creep, a few strategic choices have been made:

1. Different projects for different functions: Instead of trying to develop one large application that controls all test hardware, having many varieties that are designed specifically is desired.
2. Keep it open and accessible: Try to develop with ease of access in mind, as that is a core aspect of this project.

1.1.2 Where to start?

There is no unique good path to browse these docs. If you want to get your hands dirty and already have and AD2 and battery testing hardware, you can consider checking the *FAIR-Battery 101*.

1.1.3 How the name was chosen

The name "FAIR-Battery" is inspired by the FAIR data project which stands for Findable, Accessible, Interoperable, and Reusable. These are the principles by which the project and data it generates strive to abide by. For more information visit this link: https://www.go-fair.org/fair-principles/

1.1.4 Glossary of Terms

- AD2: The Analog Discovery 2 Board made by Digilent. In this project it is used as a hardware controller, for battery testing.
- PPS: Programmable Power Supply
- MVC: The Model, View, Controller code structure

1.2 Unboxing the FAIR-Battery [WIP]

1.2.1 Getting started

Welcome to your FAIR-Battery! Below is a list of everything you should have to get started.

Included:

- Testing Hardware

Not Included:

- Chemicals

1.3 How to Install

Currently there is only one way to install the FAIR-Battery testing software, and that is from the source. The most recent version of the project is stored on this repository.

1.3.1 Requirements

The following software is required:

- Digilent Waveforms SDK
- git
- python
- pip

1.3.2 Installation from source

Building the FAIR-Battery dependencies are tested on Windows and Mac PCs. It should be possible to install also on linux but we have not tested it yet.

```
git clone https://github.com/SanliFaez/FAIR-Battery.git
cd FAIR-Battery/
pip install -r Battery_Testing_Software/requirements.txt
```

For a more in-depth step-by-step tutorial, see *FAIR-Battery 101*.

1.3.3 Getting started

Once you have installed FAIR-Battery successfully, you can test your installation by running the *FAIR-Battery 101* project.

If you want to start editing or adding to the code, we recommend that you fork the repository first to your own account and install it from there. This way of installation allows you to stay connected with the FAIR-Battery repository and when needed, rebase to future releases.

1.4 FAIR-Battery 101

As a "Hello World' this page will be going through the basic startup process, to get you up and running as soon as possible. For general installation steps see *How to Install*.

Warning: Not yet tried on linux, although the steps should be very similar to the "For Mac" section.

1.4.1 For Mac

First you will need to install the waveforms SDK. It can be downloaded from the Digilent site.

NOTE: In the installer dragging the app to Applications is optional, but the "Waveforms SDK" MUST be added to your frameworks!

You need to have git installed first. Then in a terminal:

```
cd parent/directory/of/project
git clone https://github.com/SanliFaez/FAIR-Battery.git
```

Next you need to have python and pip installed. Verify this by running the following:

```
python3 --version
pip3 --version
```

If all is well, these will both return a version number, otherwise please install them before continuing.

NOTE: If these don't work, try using "`python`" and "`pip`" instead. If these work (and show the correct latest versions) use them instead the version with a "3" for the rest of this guide.

Next complete the installation by running:

```
cd FAIR-Battery/
pip3 install -r Battery_Testing_Software/requirements.txt
```

If this completes successfully, you should be ready to open up some testing software.

First connect the usb of the AD2, then in path FAIR-Battery/ run:

```
export PYTHONPATH=.
python3 Battery_Testing_Software/examples/101_project/BatteryTest_View.py
```

1.4.2 For Windows

First you will need to install the waveforms SDK. It can be downloaded from the Digilent site.

NOTE: In the installer choosing the app is optional, but the "Waveforms SDK" is mandatory!

Next you need to have git installed first. Then in CMD:

```
cd C:\parent\directory\of\project
git clone https://github.com/SanliFaez/FAIR-Battery.git
```

Next you need to have python and pip installed.

Make sure to check the "Add Python #.# to PATH" option when installing!

Verify this by running the following:

```
python3 --version
pip3 --version
```

If all is well, these will both return a version number, otherwise please install them before continuing.

NOTE: If these don't work, try using "`python`" and "`pip`" instead. If these work (and show the correct latest versions) use them instead the version with a "3" for the rest of this guide.

NOTE: If this isn't working after installation, perhaps python is not yet added to path.

Next complete the installation by running:

```
cd FAIR-Battery
pip3 install -r Battery_Testing_Software\requirements.txt
```

If this completes successfully, you should be ready to open up some testing software.

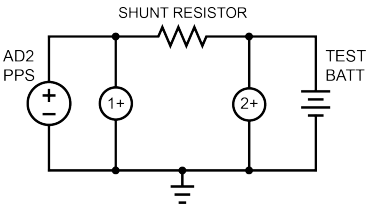
First connect the usb of the AD2, then in path FAIR-Battery/ run:

```
set PYTHONPATH=%PYTHONPATH%;.
python3 Battery_Testing_Software\examples\101_project\BatteryTest_View.py
```

1.5 Technical Drawings

1.5.1 Battery Charging Circuit

This circuit charges a battery using the AD2's programmable power supply (PPS).



The charge current is read with the two voltmeter channels of the AD2, 1+ and 2+. This allows us to get a differential voltage measurement across a known resistance, and therefore calculate the current simply:

$$I_{charge} = I_{shunt} = \frac{V_1 - V_2}{R_{shunt}}$$

Programmatic Operation

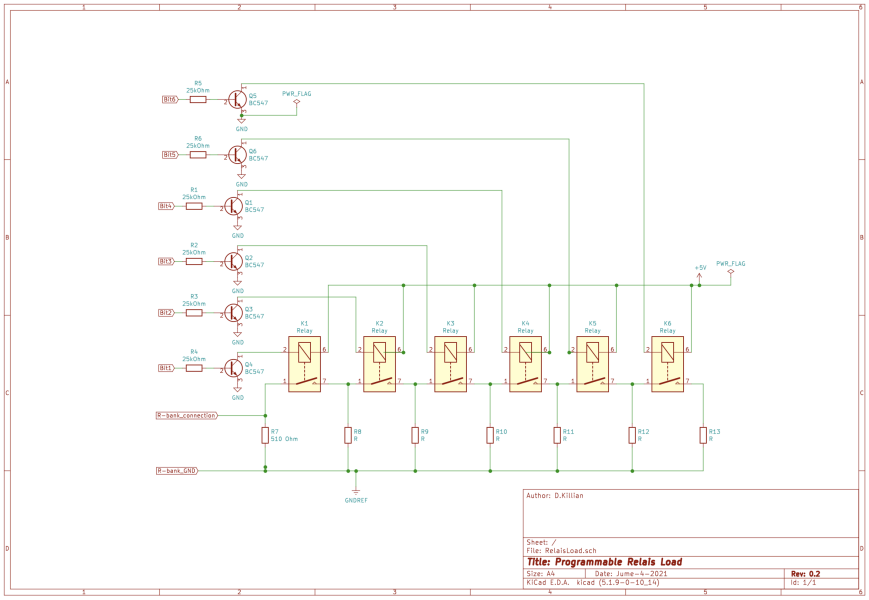
In order to use this test circuit as a constant current source, one must run a loop that adjusts the PPS output constantly to the desired current. This current is calculated using the above formula, for which a precise value of R_{shunt} must be known.

Limitations

- When not running, the battery discharges into the AD2
- Max current supply limited by R_{shunt} value and AD2's max PPS voltage of 5V
- Current measurement accuracy limited by R_{shunt} value and AD2's voltage measurement accuracy

1.5.2 Battery Discharging Circuit

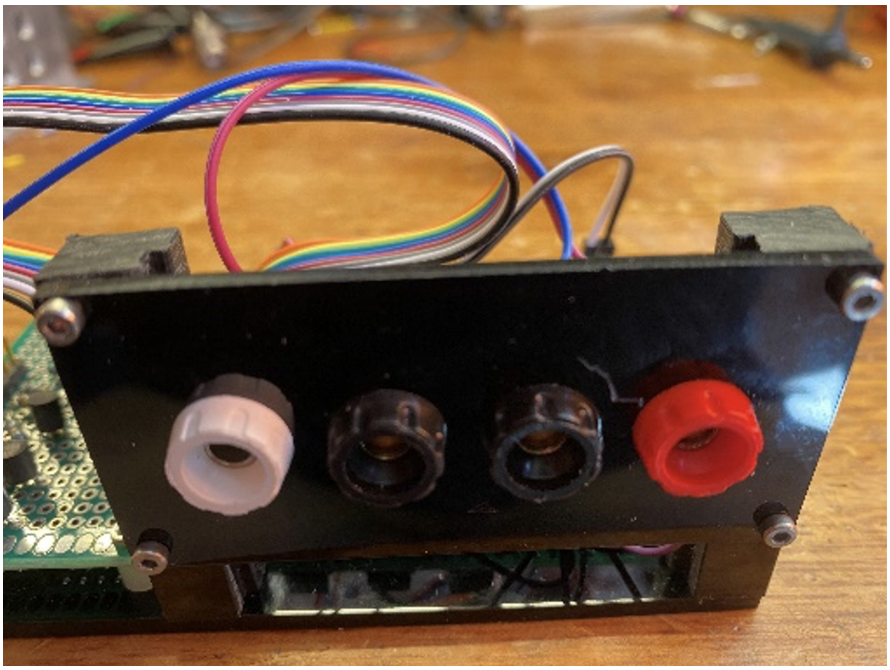
This circuit is a simple electronic load. It works by connecting resistors in parallel to achieve lower and lower resistances. These resistors are selected by relays, an electromechanical switch.



With six "bits" the load can be varied between ~500 Ω and ~10 Ω. Every extra "bit" divides the load by roughly 2.

Table 1: **Control Bits and Resulting Resistance**

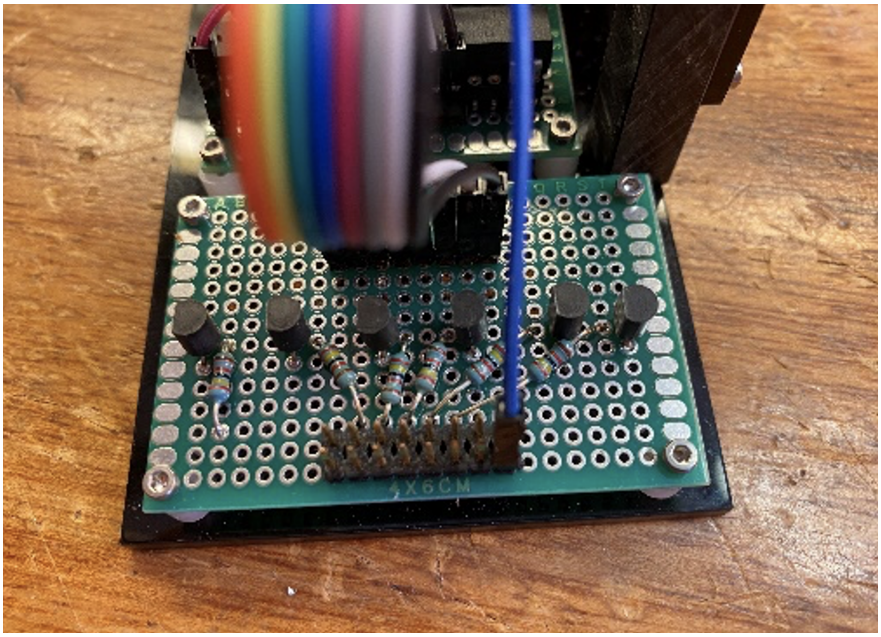
Bits ON	Load Value (Ω)
None	510
1	225
1 and 2	131
1, 2, and 3	65.9
1, 2, 3, and 4	32.9
1, 2, 3, 4, and 5	16.9
1, 2, 3, 4, 5, and 6	8.9



This little panel contains 4 banana sockets. From left to right:

- White: the connection to the load.
- Black 1: common connection for the load
- Black 2: GND for the 5 Volts supply
- Red: connection for +5 Volts

The load connections are totally separated from everything else.



Use the double row header to connect the digital signals from the AD2 to switch the load. Beware, the bottom row is all GND. The top row starts from left to right with bit 1 to 6. 7 and 8 (the two most right ones) are not connected, but can be used in the future.

Programmatic Operation

In order to use this circuit as a CR sink, simply connect the control bits according to table: *Control Bits and Resulting Resistance*. If instead a CC sink is desired, one could measure voltage drop over the load, and knowing the current set resistance, calculate the discharge current from the battery.

Limitations

- Limited number of discrete steps greatly limits smooth CC functionality.
- Max power dissipation limited by 1/4 W resistors -> Parallelled allows more, but resistances are not the same, so current is not dissipated equally

1.6 Materials Database [WIP]

One of the main objectives of the FAIR-Battery project is to accommodate alternative chemistries or materials as much as possible, and therefore remain versatile and varied.

Since a great strength of this project is the repeatability and comparability it provides redox-flow battery research, a database for doing exactly that has been created. Once a battery has been tested (see *Testing a Unit [WIP]*), the results can be submitted into the database for analysis.

1.6.1 How to Submit

Submit data by…

1.7 Testing a Unit [WIP]

Once a full cell has been assembled, it must be tested in order to compare it to other designs or chemistries. Below are some important metrics that can be used to evaluate the performance of a battery:

1.7.1 Important metrics when measuring batteries:

- Charge / Discharge at CC characteristics
- Impedance Spectroscopy
- Coulumbic efficiency
- Capacity
- Lifetime cycling
- Polarization at various current outputs
- Cyclic voltammetry

1.8 Troubleshooting

Is something not working right? Hopefully the steps below will help you get your battery up and running again, otherwise please submit an issue. See *How to contribute* for more information on how to submit an issue and what to include.

1.8.1 Hardware Troubleshooting

Troubleshooting Steps:

1. Turn it off and on
2. Check the connections
3. Check the voltage test points

1.8.2 Software Troubleshooting

Simple troubleshooting can be done by reading the popup errors that the Battery Testing software provides. More sophisticated issue however may require you to read the errors that appear in the terminal window of your code editor.

1.8.2.1 ImportError: No module named 'Battery_Testing_Software'

This occurs when your python project path variable has not been set. To fix this open a terminal and navigate to the project root directory:

```
cd FAIR-Battery/
```

Next set the path to the current directory.

Mac / Linux:

```
export PYTHONPATH=.
```

Windows:

```
set PYTHONPATH=%PYTHONPATH%;.
```

If you want to be sure the path is set correctly, you can run the following one liner in your terminal to list your paths. As ever, use either `python` or `python3` depending on which works on your system.

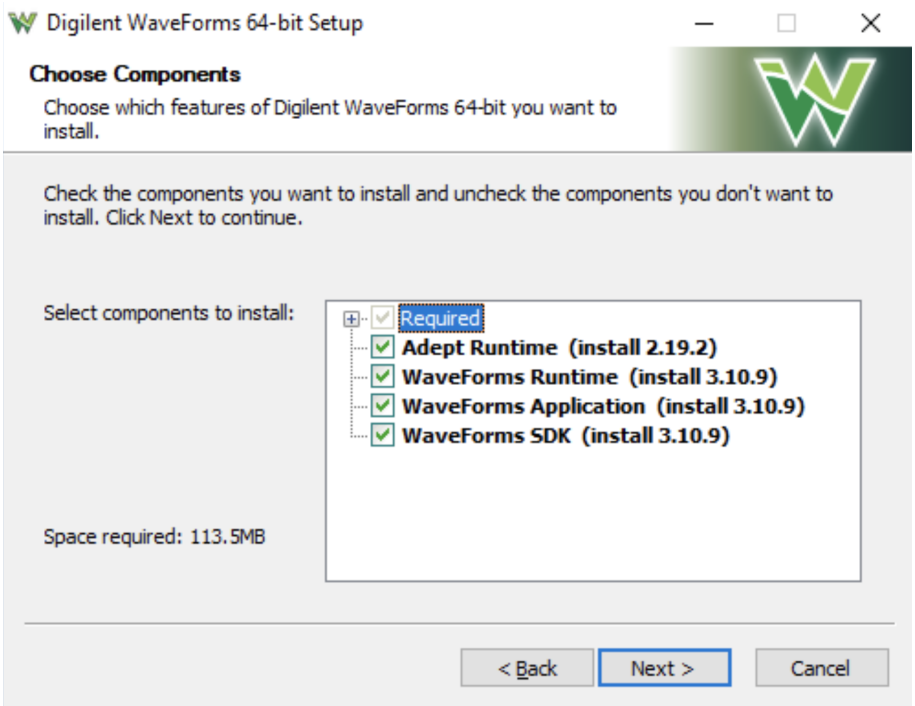
```
python -c "import sys; print(sys.path)"
```

1.8.2.2 FileNotFoundError: Could not find module 'dwf' (or one of its dependencies)

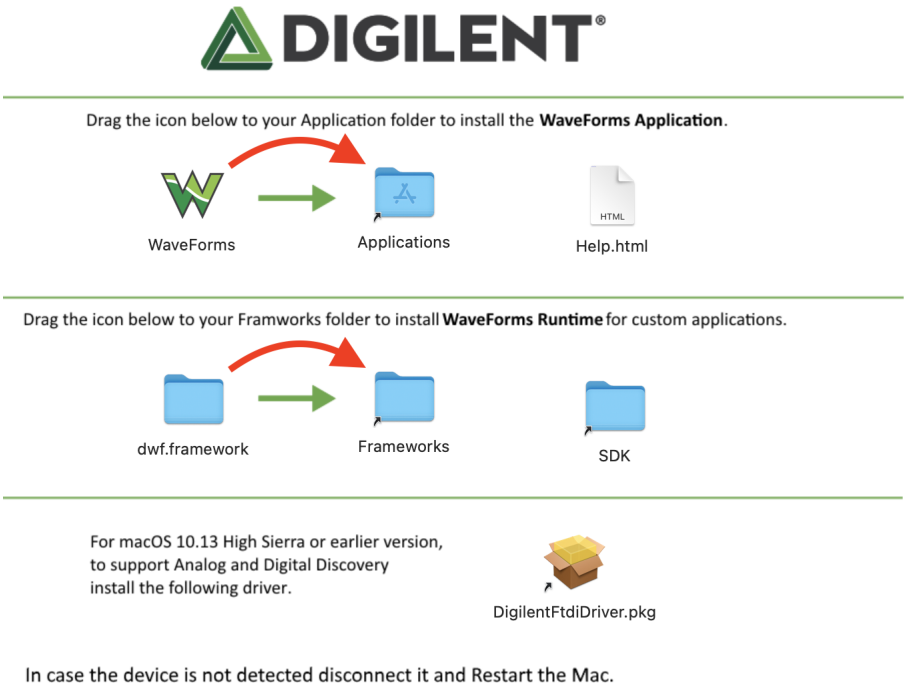
This happens when the waveforms SDK is not installed correctly. It is a dependency of the dwf module, and therefore essential for controlling the AD2. For a great startup guide visit: https://digilent.com/reference/software/waveforms/waveforms-3/getting-started-guide

NOTE: Make sure to not just install waveforms, but most importantly install the runtime and SDK.

On windows this is just a matter of checking the following options in the installer.



On mac once the installer is opened, drag not only the waveforms app to applications, but also the dwf.framework to your Frameworks folder.



1.8.2.3 Error: AD2 Device not connected

Check the USB connection to the Analog Discovery 2 board. Try various USB cables in case one has broken or does not have data lines. Perhaps also try the same USB cable with another device that requires data lines, to make sure this isn't the issue. If all as well, the AD2 should appear as a USB device.

1.9 Customization

While one of the goals of this project is to reduce variation in testing methods, this software and hardware is still open source, and very open to improvement!

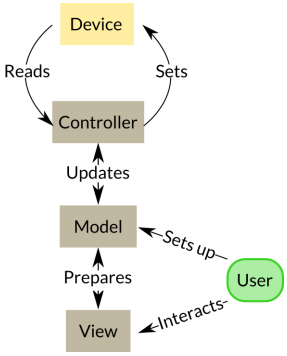
Every measurement setup of protocol has its own logic and requirements. Therefore, unless you have very basic requirements for your measurements, labphew is meant to be hacked/customized/edited before is can match your needs. If you are an experienced python programmer, you can customize labphew by editing the code at any level you need. Hopefully, there is enough value in the current code that helps you not to start from a blank page.

If you are still in the learning phase, or would like to stay compatible with the existing modules of labphew these recommendations might help you in getting your desired features rather quickly.

1.9.1 Software Structure

In order to get you started quickly, below is a quick summary of how the code is structured. For a more in depth explanation of how this project's software is structured, see the labphew documentation.

Summarized, the the software follows a Model, View, Controller structure.



In this structure the code is separated into different subfolders, with a clear separation of tasks. These modules and their interactions with the user and electronics are schematically presented in the figure above. The MVC subfolders can be explained as follows:

- *model*: Specifies the logic and the order of operations to perform a certain measurement.
- *view*: User-facing elements, such as the graphical user interface (GUI), or the command-line interface (CLI).
- *controller*: The drivers for the hardware. These modules handle the low-level communication with the devices.

Additional to the MVC subfolders, labphew has three subfolder for managing communication with users

- *core*: Includes the templates and base classes that are introduced for inter-operability.
- *docs*: Contains the documentation, such as this file
- *Examples*: a few examples of labphew applications written for generic or special purposes

To further understand the architecture of the code, we specify some of the main contents of each module.

1.9.1.1 Controller

The controller can be regarded as the driver for the device, or the Python wrappers for the driver. For the AD2 this is: `labphew/controller/waveforms:`

The waveforms controller (DfwController) is a module that implements many of the the basic functionalities for controlling the AD2 board, inherited from the dwf library provided by Digilent. Examples include setting analog outputs channels or reading from oscilloscope input channels. Many of these features have been implemented, but the controller is not complete, so this could be expanded upon if needed.

1.9.1.2 Model

Models translate the logic of the measurement into machine-iterable algorithms that communicate with the devices using the controller module. Each model contains an `Operator` class (remember the Operator in the Matrix series?). A simple example could be capturing a single frame with a camera in which it is usually needed to open a shutter, trigger the camera, acquire a frame, close the shutter, and readout the acquired data and save or display. This specific sequence of operation are determined by the user and thus should be separated from the controllers. In labphew, this procedure is included in various method of the Operator class, for example `analog_out` which controls the voltage on the signal generator output, or `pps_out` which sets the voltage of the programmable power supply.

1.9.1.3 View

The view folder is the collection of interfaces for users to run an experiment. Different views can be written for the same model, dependent on how the user likes to interact with the setup. A generic view can also work with different models. A view module is usually initiated by calling an `Operator` to take care of the communication with devices and the steps of the main control loop. Views can be both working with command-line interface (CLI) or a graphical user interface (GUI).

An example of this is the `BatteryTest_View.py` which has been developed to implement various basic battery testing functions.

1.9.1.4 Resources

- Waveforms SDK
- AD2 Specs

1.10 How to contribute

We are currently testing and troubleshooting the first beta release. There is no written roadmap, or any long-term plan except for fixing all the glitches in the current code. If you see room for improvement and would like to scratch your itch, please feel free to contact Sanli Faez and get involved in making this project grow.

Other small issues can be reported via the issue tracker

1.10.1 Adding examples

If you have customized FAIR-Battery Software for your own project, please consider stripping your additional code in to a simplified example that can be added to the examples.

1.11 Acknowledgement

- labphew - this project is a fork of labphew

If you know of a package that should be added to this list, please do not hesitate to drop us a message.