System Setup for Measurement

Charlie G. Tolley[1,2]

[1]*Department of Astronomy, University of California Berkeley, Berkeley, CA 94720, USA*
[2]*Radio Astronomy Laboratory, University of California Berkeley, Berkeley, CA 94720, USA*

(Dated: Oct 1, 2025)

1. HARDWARE SETUP

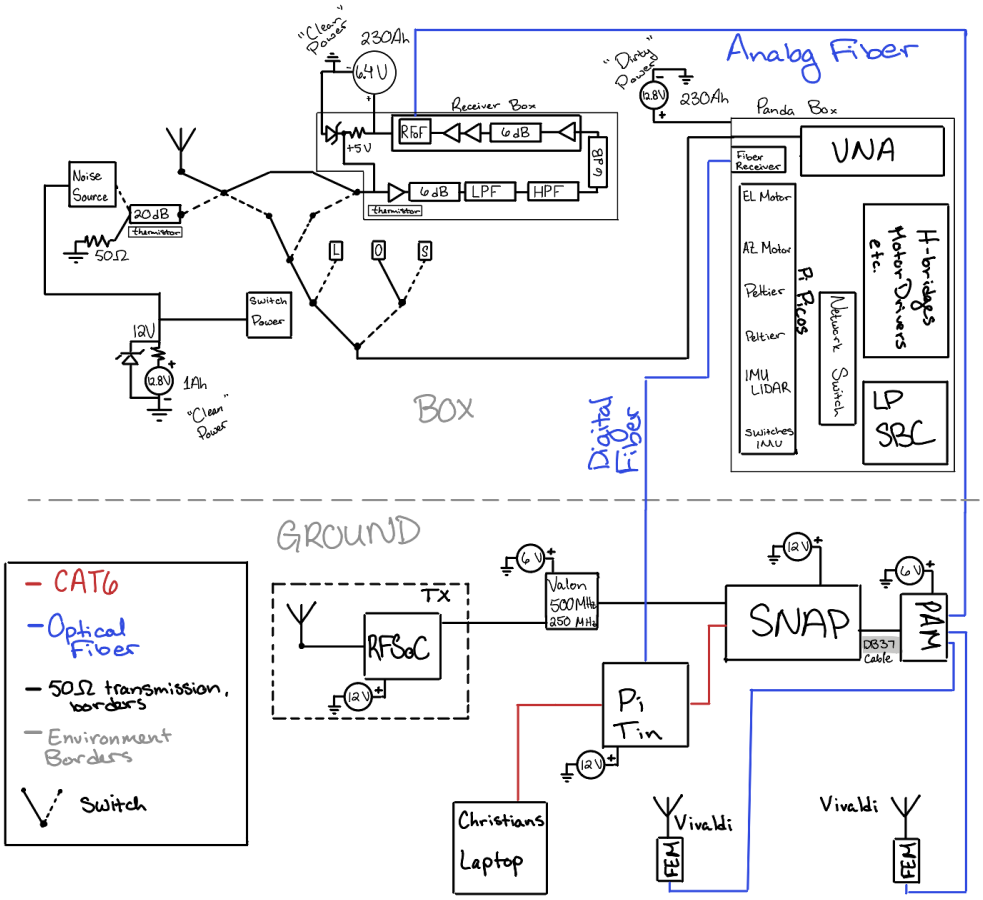

Figure 1. The configuration of the most basic components of the telescope. The only elements that are not included are the IMU, LIDAR, and Peltier devices (thermal control). Note that the DB9 cables are not pictured here - all cables in the box are labeled with their destination DB9 connector. This being said, the only cables needed for basic switching/calibration cycle within the box are the RF Switch DB9, power to the switches, noise source, receiver and panda box. Note also that the Vivaldi feeds are only included for the sake of completion, as they are only used for RFI flagging and unnecessary for the internal calibration cycle. Likewise, the RFSoC external transmitter (TX box) is not required and not setup in the lab currently.

Corresponding author: Charlie G. Tolley
tolley412@berkeley.edu

1.1. *Power*

In the lab, the systems shown can be powered by either battery or power supplies. Generally, I use power supplies for the backend (everything in the "GROUND" section of the diagram), and batteries for the box. There are three battery packs that power the box:

1. 1Ah 12V clean power (attached to the roof of the box)
2. 230Ah 12V dirty power (the big battery with the N-type cable, has an auto-reset circuit breaker)
3. 230Ah 6V clean power (big battery with SMA cable, has an auto-reset circuit breaker)

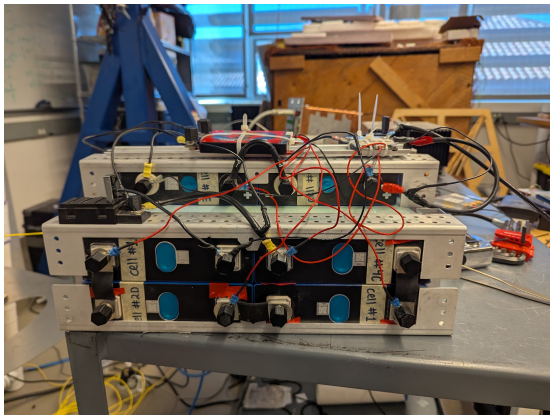

Figure 2. Front view of the 230Ah batteries.

The clean box components all have zener diodes connected to regulate voltage as our batteries are not truly 3V per cell, but closer to 3.2V per cell. To ensure that the voltage levels of the LNAs (which have voltage and temp dependencies) remain consistent between the power supplies and batteries, make sure to set the voltage levels accordingly.

The thermistors, which provide thermal monitoring, are powered through connection to the Pi Pico microcontrollers and do not require an external power source.

1.2. *The Front-end*

The front-end box (see Fig 3) is generally always set up in the box - at the most, you may need to reconnect the optical fiber to the frontend output port, the 6V power to the power port, or the RF input port. All of these ports and cables are labeled or self-evident by connector type. In broad strokes, the switches connect all possible DUTs to the receiver, uses 6V to power an LNA and the RFoF board based off Kiran's design and modified by Wei (ftx, frx), and outputs an optical signal for low-loss transmission across our 100m air gap.

Figure 3. Front-end box in telescope box.

1.3. *The Back End*

The backend is composed of the Post-Amplification Modules (PAMs), which receive our analog RF signal as optical signal, the Smart Network ADC Processor (SNAP), the Pi Tin, and the Valon reference clock (see Fig 4). You will have to be aware of the SNAP that you are using. In the current setup, the HERA number is `C000069`, and the other SNAP we have available is `C000122`. This has implications for which `corr_config.yaml` you use when initializing the FPGA. For the PAMs, it doesn't matter which fiber input/SMA output port pair you use, but make sure to clean the fiber port. To connect to the Pi tin (which talks to the LattePanda SBC), one can connect via wireless connection or a wired connection, which is shown in Fig 1. The Valon 500MHz clock connects to the Sample Clock port on the SNAP.

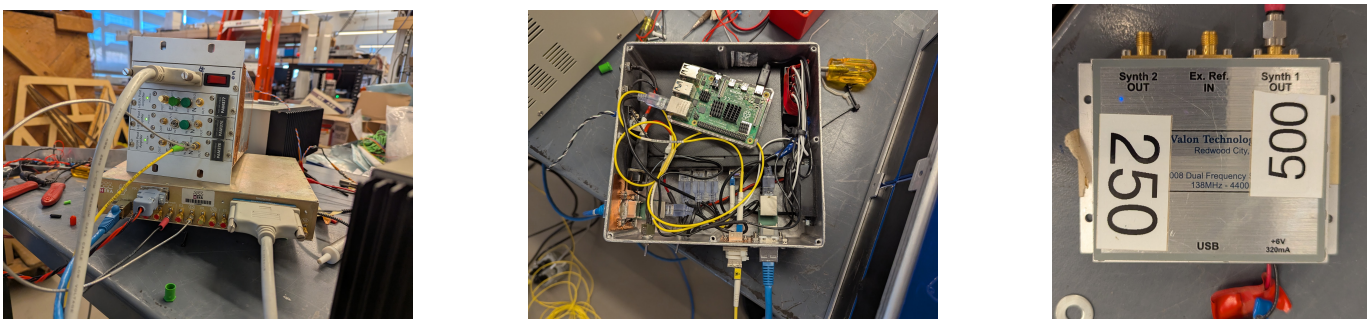

Figure 4. PAM and SNAP (left), Pi Tin (center), Valon Reference Clock (right)

2. SOFTWARE

There's a couple of ways to take measurements, both of which I'll touch on in this section:

1. Manually switching through and taking measurements.

2. The observing scripts used in the field.

2.1. *Manual Access/Measurements*

Accessing the system manually entails initializing the SNAP board, running a live plotter (optional but useful), a terminal to take a specified number of spectra, and a terminal to operate switches, VNA, and any other of the Pico functionalities. This is mostly useful for troubleshooting.

2.1.1. *Terminal 1*

Initialize the FPGA from the Raspberry Pi.

```
ssh eigsep@10.10.10.10 (requires password)
(Pi) python ~/eigsep/eigsep_observing/scripts/fpga_init.py -pasf
```

You may need to specify the config, as mentioned above. The default file is `corr_config.yaml` and has the correct information for SNAP `C000069`. If you need to switch to SNAP `C000122`, you will have to specify the correct config file with argument `--config_file ~/eigsep/eigsep_observing/src/eigsep_observing/config/corr_config_snap122.yaml`.

All you need control of are switches and VNA, but thermistor control is highly recommended.

2.1.2. *Terminal 2*

Live plotter for real time monitoring. Optional, but useful, and can be run from your local server.

```
bash
    python <path to eigsep_observing>/scripts/live_plotter.py --pairs 1
```

The `pairs` flag is optional, but without it, it will plot all the outputs, including cross-correlations, which is slow. As the system is currently set up, the data we care about is coming out of "pair" 1.

2.1.3. *Terminal 3*

Connect to the box, and initialize Pico objects.

```
bash
    ssh eigsep@10.10.10.10 (RPi, requires password)
    (Pi) ssh eigsep@10.10.10.11 (LattePanda, requires password)
(Panda) python
    from picohost import PicoRFSwitch, PicoDevice, PicoMotor
    from cmt_vna import VNA

    sw = PicoRFSwitch('/dev/ttyACM2') #Check the number
    c = PicoMotor('/dev/ttyACM5')
    vna = VNA(switch_network=sw)
    #The functionality of these are documented in code
```

2.1.4. *Terminal 4*

For capturing spectra, this can also be run from local server. The following command will produce an npz file and can be run at any point.

```
    (Local) python <path to eigsep_observing>/scripts/capture_spectrum.py <# of 1 sec integrations>
    <filename>  --pairs 1
```

2.2. *Field Measurements*

Setting up data collection as performed in the field requires some preparation: of the pico config file that defines what picos you use, and of the config file that defines your data collection cycle. The Raspberry Pi runs an `observe.py` file that uploads that high-level cycle-defining config to the redis server shared by the Panda and Pi, and monitors the redis server for data that the Panda and SNAP board deposit. The Panda runs a `panda_observe.py` script that monitors the redis for commands from the Pi and deposits data it collects there.

The pico config file may need to be generated or overwritten with the `flash_picos.py` script on the panda board, which identifies what each of the picos connect to and writes it to the config json: `python flash_picos.py --output <filename> --port <serial port to generate>`. The default file that the observing script references and that `flash_picos.py` outputs to, `/home/eigsep/eigsep/pico-firmware/pico_config.json`, defines all the pico ports. For an observing run that only cares about switching and nothing else (like thermal control or motors), a `rfsw_config.json` file might show the following:

```
    [
  {
    "sensor_name": "rfswitch",
    "status": "update",
    "app_id": 5,
    "sw_state": 0,
    "port": "/dev/ttyACM2",
    "usb_serial": "7332A76E0B1C62D6"
  }
]
```

And was generated by:

```
(Panda) python flash_picos.py --output rfsw_config.json --port /dev/ttyACM2
```

This in turn is referenced by the correlation config yaml file on the Raspberry Pi in the Pi Tin, uploaded to the redis and read by the Panda from the redis. An example of a config file meant for only calibration data that does not read any sky spectra is shown below:

```
# High-level configuration for EIGSEP observations
# Sets switch schedules, picos, VNA settings, and other parameters

# IP addresses
rpi_ip: "10.10.10.10"
panda_ip: "10.10.10.11"
# picos
pico_config_file: "/home/eigsep/eigsep/pico-firmware/rfsw_config.json"
pico_app_mapping:  # app number to device mapping
  0: "motor"
  1: "peltier"
  2: "therm"
  3: "imu"
  4: "lidar"
  5: "switch"
# corr filewriter
corr_save_dir: "/home/eigsep/eigsep/cal_data"
corr_ntimes: 240
# switches
use_switches: true
switch_schedule:  # seconds per measurement
```

```
  RFANT: 0 # sky
  RFNOFF: 120  # load
  RFNON: 120  # noise source
# vna
use_vna: true
vna_interval: 60 # seconds between VNA measurements
vna_ip: "127.0.0.1"
vna_port: 5025
vna_timeout: 1000  # in seconds
vna_settings:
  fstart: 1000000.0  # in Hz
  fstop: 250000000.0  # in Hz
  npoints: 1000
  ifbw: 100.0  # in Hz
  power_dBm:
    ant: 0.0
    rec: -40.0
vna_save_dir: "/home/eigsep/eigsep/cal_data"
```

Important features include the `pico_config_file`, which points to the `rfsw_config.json` file. The directory in which the correlation files and vna files are saved, and the `use_switches` and `use_vna` boolean options. This was adapted from the observing config yaml file, which is in the same folder and only changes the time spent on sky.

See below a breakdown of the steps to run these measurements. The first two steps are the same as in the manual setup: to initialize the SNAP board, and to start up the live plotter (optional, but useful).

2.2.1. *Terminal 3*

Run the observe script from the "ground".

```
(Pi) python ~/eigsep/eigsep_observing/scripts/observe.py
--cfg_file ~/eigsep/eigsep_observing/src/eigsep_observing/config/<your yaml config file>
```

2.2.2. *Terminal 4*

Run the panda observing script from the "air".

```
(Panda) python /home/eigsep/eigsep/eigsep_observing/scripts/panda_observe.py
```

If this works, you should see scrolling log messages that the Panda client has connected to the redis and that the control, switch and vna threads have started. On the Panda terminal, any switching should showing up in the logs as well.