|  |  |
| --- | --- |
| Operating System | macOS Montery version 12.4  Linux bookworm 64-bit (raspberry pi 4 model-b) |
| Power Meter | Newport 1830-C |
| Monochromator | Orion Cornerstone 130 1/8m |

Table of Contents

[General Setup 1](#_Toc150155600)

[Installing the Driver 1](#_Toc150155601)

[Testing Connection 2](#_Toc150155602)

[Terminal 2](#_Toc150155603)

[vscode 3](#_Toc150155604)

[Power Meter 3](#_Toc150155605)

[Setup 3](#_Toc150155606)

[Documentation 4](#_Toc150155607)

[Monochromator 4](#_Toc150155608)

# General Setup

Both the monochromator and the Power meter use a RS-232 serial connection to talk to the computer. Getting it working can be a bit of a pain, so this document is to help set it up.

## Installing the Driver

The first part is installing the right driver. This can be confusing as the physical cable sometimes have a different manufacturer than the chip inside. It is important to get the driver for the **CHIP** itself.

To find out what driver you need on mac you can go to **System Information** then on the left under “Hardware” select USB. Then find the cable it should be called something like “USB-Serial Controller” Then under the information window we can see both the Product ID (PID) and Vendor ID (VID).

A computer screen shot of a computer

Description automatically generated

In my case the Vendor (ATEN) sells the cable but the actual chip inside is made by **Prolific Technology inc.** Go to the manufactures website, download and install the most recent version of the driver.

\*\*\* If you have any other drivers installed it may not work so make sure to check and delete other drivers \*\*\*

## Testing Connection

Now that the right driver has been installed a quick check should be done to ensure proper operation. An easy way of doing this is by shorting the transmit and receive data pins together – this will echo back any characters sent out from the computer back to itself. The transmit and receive lines are on pins 2 and 3 – they are mirrored depending on if It is a male/female connection so just double check using the photo below:

A diagram of a male and female connection

Description automatically generated

To send data there are two ways – either through using screen in terminal or using the serial monitor extension is vscode the ladder of the which is a little simpler.

There are two types of serial cables -> pass through and crossover. FOR BOTH THE MONOCHROMATOR AND THE POWER METER YOU NEED A PASSTHROUGH CABLE– most null modem cables wont work.

### Terminal

Open up a terminal window and type:

$ screen /dev/tty.usb

Then press tab. It should autocomplete and fill in the right port – mine looks like:

$ screen /dev/tty.usbserial-1140

In the new screen window that opens up just type a character and hit enter and you should see it echo back in the terminal which confirms the setup is working. (Note when typing commands into screen what you are typing will not be shown)

### vscode

In extensions install the “Serial Monitor” extension by Microsoft. Open up the terminal in vscode and switch to the “Serial Monitor” tab. Under port select the tty.usbserial-XXX device and then type a character into the prompt at the bottom and confirm it is echoed back.

**THIS IS ALSO PROBABLY THE EASIEST WAY TO FIND OUT WHAT PORT THE SERIAL DEVICE IS CONNECTED TO.**

# Power Meter

This section goes into detail setting up and using the power meter.

## Setup

On the back of the power meter the switches need be set according to the figure below. The baud rate should be kept at **9600.**

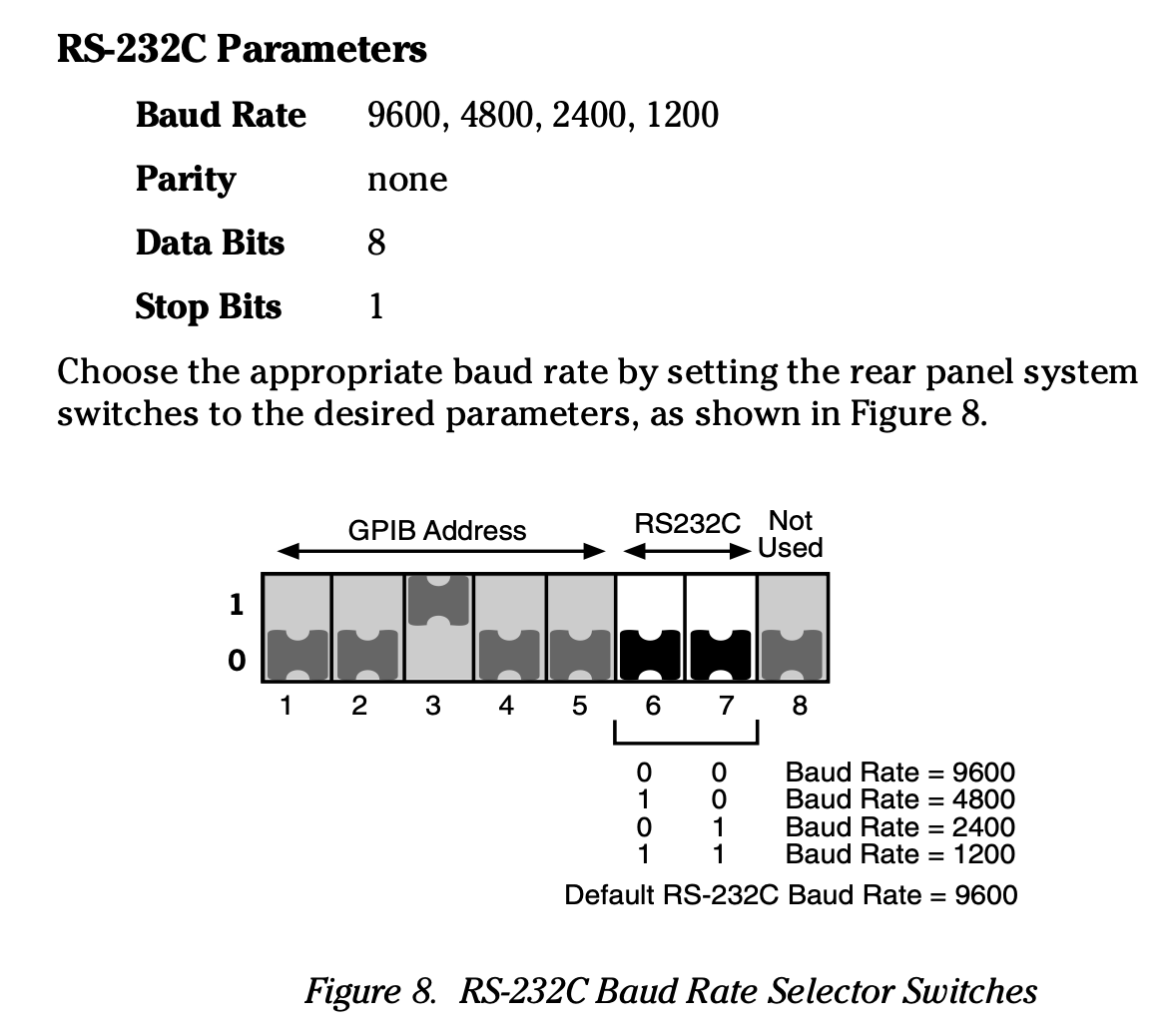


Figure 1. Power Meter Switch Settings

A **passthrough** cable should be connected to the Power Meter from the USB-serial adapter. A null modem cable will **NOT** work.

## Documentation

Controlling the power meter is implemented as a class in **power\_meter.py**

There are many different functions apart of the class but you can get away with only using two:

1. write()
2. writeAndRead()

The first method: write() writes a command to the power meter. Use this method when sending a query that does not require a response. For example, setting the wavelength of the power meter. The command you are sending must be formatted as a string.

The second method: writeAndRead() writes a command and returns the response. This is useful for sending a query that you want the response of, for example getting a reading. (Although there is a dedicated readData() method for this purpose). The command you are sending must be formatted as a string.

It is important to keep track of what commands send back a response as they are not cleared from the buffer until they are read so sending a query and not reading a response means the old response will be read in any subsequent calls.

A full list of commands to use with the write()/writeAndRead() method is available starting on page 27 of the power meters manual Example usage looks like:

|  |
| --- |
| import Photodiode  import time  # Create photodiode object  photodiode = Photodiode()  # Set wavelength to 400  reading = photodiode.write("W400")  # Sleep for one second  time.sleep(1)  # Send wavelength Query  wavelength = photodiode.writeAndRead("W?")  # Print wavelength setting  print(reading) |

There are additional methods which are documented in the Photodiode.md file in the repo.

# Monochromator

Controlling the monochromator is implemented as a class in Monochromator.py

Everything is similar to the power meter class with one big caveat. The monochromator echo’s every command back and I do not believe you can turn this off like with the power meter. This means every command needs to be read to clear the buffer after executing. This means the write() method also includes a read. And then the writeAndRead() method has two reads, one to clear the echo and the next to get the actual response.

Most likely the only method that you will be using is the **goWave()** method which takes a integer wavelength (in nm) and moves the monochromator to that. An example on how to use it is below:

|  |
| --- |
| import Monochromator  import time  # Create Monochromator object  m = Monochromator()  # Sweep through visible light (400nm - 600nm)  for wl in range(400, 600, 50):  # Move monochromator  m.goWave(wl)  # Wait 1s  time.sleep(1) |