Point One Standard Dev Kit

AH (Heading) Firmware

Application Note

V1.3

6/16/2023

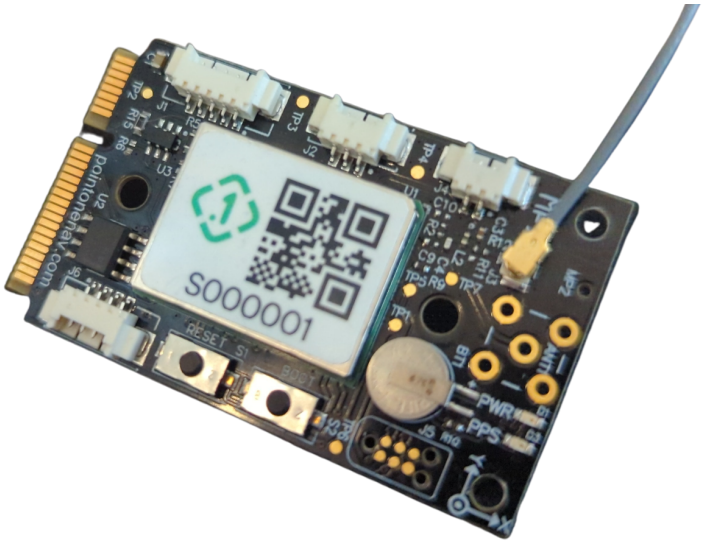
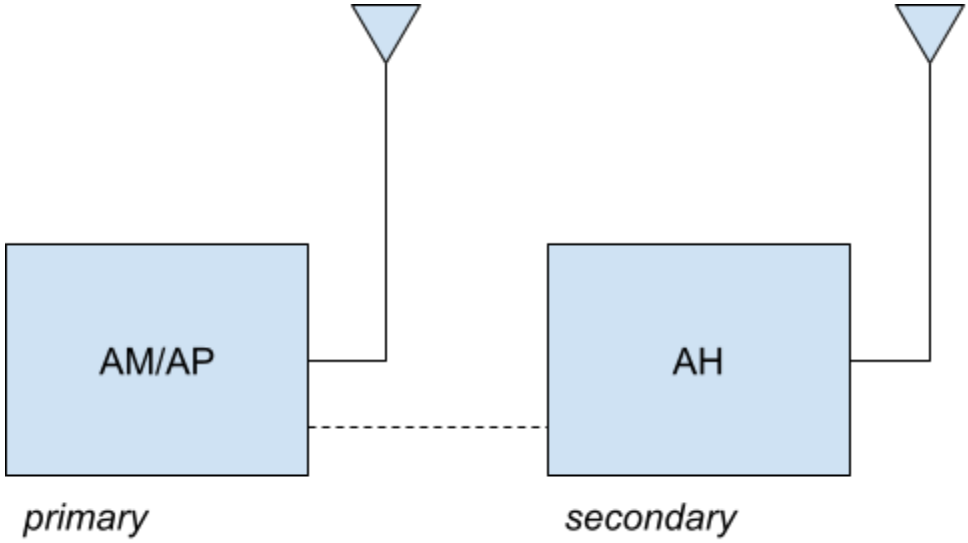


Table of Contents

1. Introduction

This document includes hardware and operating details specific to the Point One Standard Development Kit (P1SDK) AH (heading) firmware. It is an addendum to the Point One Standard Development Kit (P1SDK) User Manual. It is recommended to completely read through the User Manual for background information and operating instructions for the dev kit before following this document.

This document provides technical guidance for using the Point One Standard Dev Kit with the **AH** firmware to act as a heading measurement engine. This firmware allows an AH *secondary* device and GNSS antenna to be used in conjunction with either AM (GNSS-only) or AP (INS) navigation engine firmware on a *primary* device in order to determine an orientation solution independent of platform motion.



During operation, the secondary AH device will produce a 10 Hz FusionEngine HeadingMeasurement message (11001), detailed in the Point One FusionEngine protocol (https://github.com/PointOneNav/fusion-engine-client).The output message will be available from the *primary* device. This allows for an architecture where the secondary device is only connected to the primary device if desired.

2. Requirements

- Hardware
  - 2x Point One Standard Dev Kit (with carrier board) or Quectel EVK
  - 2x L1/L5 GNSS antenna
- Software
  - Standard Dev Kit AH release package containing the latest firmware and tools
    - Latest firmware version available at https://pointonenav.com/docs/#standard-dev-kit
  - Windows, Linux (kernel v5.9 or later), or Mac OS

- Connectivity
  - The primary module can be connected over USB/Serial to a host computer
  - The secondary module can be connected over USB/Serial to a host computer or directly to the primary module
  - It is highly recommended to include at least one connection from the secondary module to the host computer to facilitate software updates in the field

NOTE: DRIVERS ARE REQUIRED FOR WINDOWS AND MACOS

When using Windows, install the CP210x Windows Drivers v6.7.6 or later from https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers. The default universal driver included with Windows can cause significant unexpected data loss on some machines.

When using Mac, install the CP210x VCP Mac OSX Driver v6.0.0 or later from https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers.

When using Linux, use the CP210x driver included with Linux kernel version 5.9 or later (included in Ubuntu 21.04 and later). Before 5.9, the driver had an issue that can cause unexpected data loss on some machines.

3. Hardware Configuration

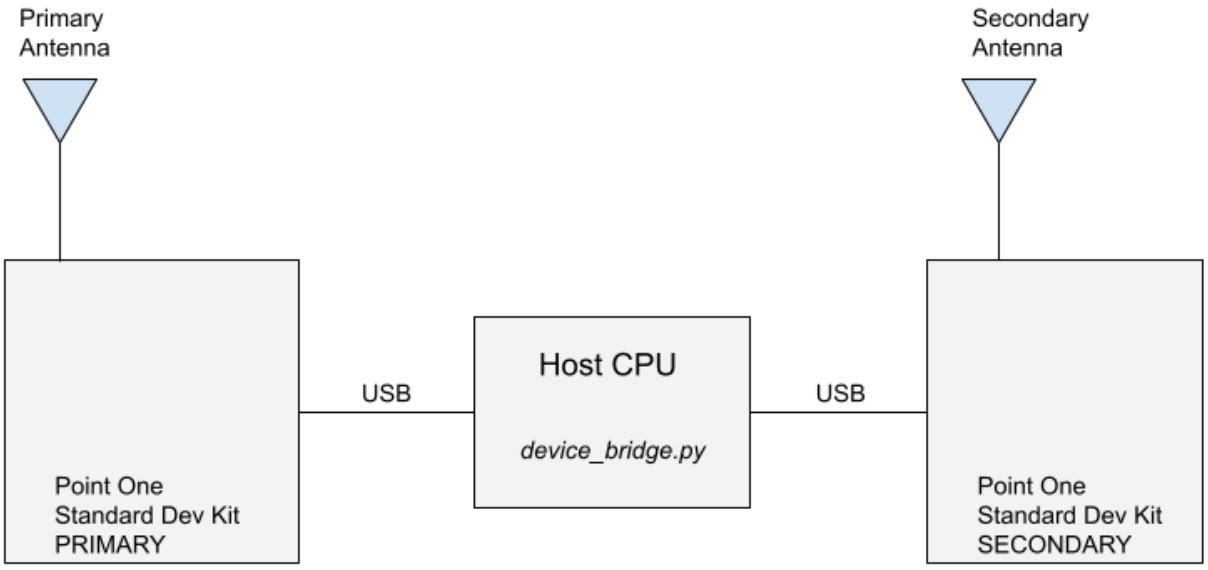
3.1 Installation Considerations

A complete heading solution consists of two devices: a *primary* navigation engine and *secondary* heading measurement engine. Consider the following recommendations when setting up your equipment:

- The primary and secondary antennas should have a minimum of 0.5 meter distance baseline. Greater distances between the antennas result in higher angular accuracy.
- The angle is reported as measured *from the primary antenna to the secondary antenna.*
- It is recommended to use the same type of antenna for both the primary and secondary if possible.

Note: The setup guide for the Quectel EVK (large green circuit boards) are in Appendix B.

3.2 Configuring Standard Dev Kit Hardware for Heading



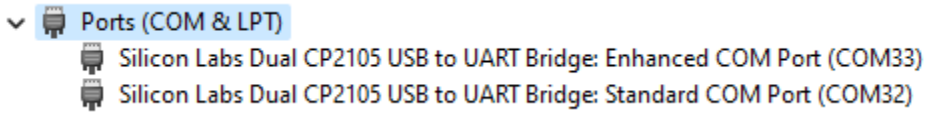
When using the Point One Standard Dev Kit, the recommended configuration is to plug both devices via USB (or directly via mPCIE) into a host computer. Each Dev Kit will enumerate on the computer as two serial ports, resulting in a total of four new serial ports on the host computer.

Note that for this setup, corrections must only be applied to the primary device. The secondary device doesn't need corrections applied as once the device bridge is created, the secondary device receives data from the primary device.

3.2.1 Identifying UART Ports

After connecting the devices, the serial port numbers assigned by the host computer for each of the UARTs on the two devices (total of 4 UARTs).

In Windows, you can retrieve the COM port for your device by opening Windows Device Manager and expanding the Ports (COM & LPT) Menu:



On Linux or Mac, you can open a terminal and use the `ls` command to look for devices that match the `/dev/ttyUSB*` pattern (or `/dev/tty.SLAB_*` on Mac):

Linux: `ls /dev/ttyUSB*`

Mac: `ls /dev/ttySLAB_USB*`

Note that the `/dev/ttyUSB*` or `COM*` numbers may change when the device is unplugged. On Linux machines, it is highly recommended that you create a udev rule to ensure a consistent enumeration of the devices in order to prevent confusion between the two devices, and with other USB to Serial devices you may have. See Appendix C for details.

On most host computers, the first connected device will enumerate as follows:

	Windows	**Linux**	**Mac**
UART1	Standard COM Port	`/dev/ttyUSB1`	`/dev/tty.SLAB_USBtoUART1`
UART2	Enhanced COM Port	`/dev/ttyUSB0`	`/dev/tty.SLAB_USBtoUART`

NOTE: This mapping only applies for the Point One Standard Dev Kit. It DOES NOT apply for the Quectel EVK. See the Point One Standard Dev Kit User Manual for details.

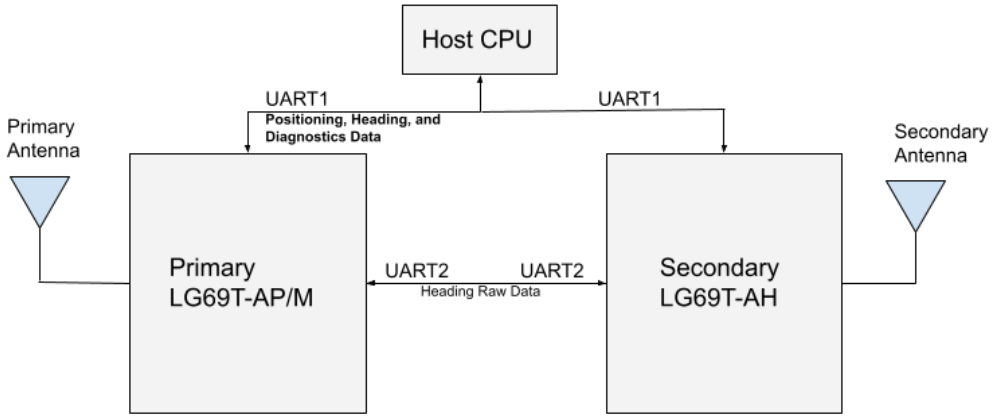
For the heading function to work properly, data from the primary UART2 must be routed to the secondary UART2.

Note: The following table includes an example serial port numbering. The commands in the sections below assume this configuration. Make sure to write down the correct serial port names on your particular host PC and use those values in the examples below.

	Windows	Linux	Mac
Primary Device UART1	Standard COM Port COM1	`/dev/ttyUSB1`	`/dev/tty.SLAB_USBtoUART1`
Primary Device UART2	Enhanced COM Port COM2	`/dev/ttyUSB0`	`/dev/tty.SLAB_USBtoUART`
Secondary Device UART1	Standard COM Port COM3	`/dev/ttyUSB3`	`/dev/tty.SLAB_USBtoUART3`
Secondary Device UART2	Enhanced COM Port COM4	`/dev/ttyUSB2`	`/dev/tty.SLAB_USBtoUART2`

3.2.2 Configuring the Modules

The default AM,AP, and AH firmwares come pre configured for heading applications, if using the configuration laid out below:



If communicating via UART1 to the host computer like shown above, you may want to set diagnostics on for troubleshooting, and to turn on all additional FusionEngine messages besides heading by using the commands below:

```
./bin/config_tool.py --device-port /dev/ttyUSB1 apply uart1 message_rate fe all on -f
./bin/config_tool.py --device-port /dev/ttyUSB1 apply uart1_diagnostics_enabled true
./bin/config_tool.py --device-port /dev/ttyUSB1 save
```

3.2.3 Linking the UART Ports

Now that you have identified your devices, you can use the `device_bridge.py` application (included in [p1-host-tools github]) to easily link your Primary and Secondary devices.

Windows: `device_bridge.exe COM2 COM4`
Linux: `python3 device_bridge.py /dev/ttyUSB2 /dev/ttyUSB0`

Remember that you should change `COM2` or `/dev/ttyUSB0` above with the value assigned to your Primary UART2, and you should replace `COM4` or `/dev/ttyUSB2` with the value assigned to your Secondary UART 2.

See `python3 bin/device_bridge.py --help` for additional options and details.

The only requirement is that data is forwarded between Primary UART2 and Secondary UART2 in both directions with minimal data loss and low latency, ideally under 10 milliseconds. Note that `device_bridge.py` is an example program. It can be extended to bridge devices over two host computers over an Ethernet link (TCP/UDP) or over CAN if desired. Additionally, the boards can be directly wired if desired.

Congratulations, you have successfully configured the devices for heading operation. This will continue to run in the background while you are validating functionality and need devices to communicate. See Section 4 for working with the Point One Desktop Application to verify correct installation.

4. Verifying Functionality Using Software for Heading

After following Section 3.2.3 to enable the FusionEngine HeadingMeasurement output message, you can easily verify heading operation using either the Point One Desktop Application (available at https://pointonenav.com/docs), or using the `p1_display` and related analysis tools in the FusionEngine repository (https://github.com/PointOneNav/fusion-engine-client).

4.1 Using Point One Desktop Application

1. Ensure you have followed Section 3 to configure your Primary device correctly.
2. Configure the Desktop Application according to Section 5 of the Point One Standard Dev Kit User Manual and connect to your Primary device via UART1 and apply corrections.
3. Repeat step 2 to connect to your Secondary device via UART1, but **DO NOT** apply corrections.
4. In an open sky environment, you should see a “Heading” section appear in the tray as indicated by the yellow arrow below while looking at the Primary device. This heading display shows the output of the dual antenna heading solution computed by the AH module.

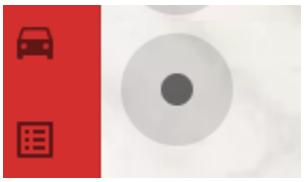
4.2 Using p1_runner

1. Similar to above, only the primary needs to have corrections enabled, either via NTRIP or Polaris API
2. Ensure that you are still running device_bridge.py script from Section 3.2.3
3. To log and your navigation using p1_runner, run command below:
    a. NTRIP:
        i. `./bin/runner.py --device-port /dev/ttyUSB1 --ntrip http://169.125.0.1:2101,my_mountpoint,my_username,my_password`
    b. Polaris:
        i. `./bin/runner.py --device-port /dev/ttyUSB1 --polaris abcd1234`
4. User can log secondary device(via UART1) if desired, but not required

4.3 Using Point One FusionEngine Client Analysis Tools

All Point One devices run the FusionEngine core libraries. One of the benefits of this library is that it comes with free, open source tools to inspect the performance of the engine by analyzing a log file. The repository used here can be downloaded from https://github.com/PointOneNav/fusion-engine-client, or can be installed into your Python environment using pip. See https://github.com/PointOneNav/fusion-engine-client#python for details.

In order to analyze a log from Point One Desktop, you must first record a log file from the ***primary*** device. This can be done by simply pressing the record button in the Point One Desktop application:



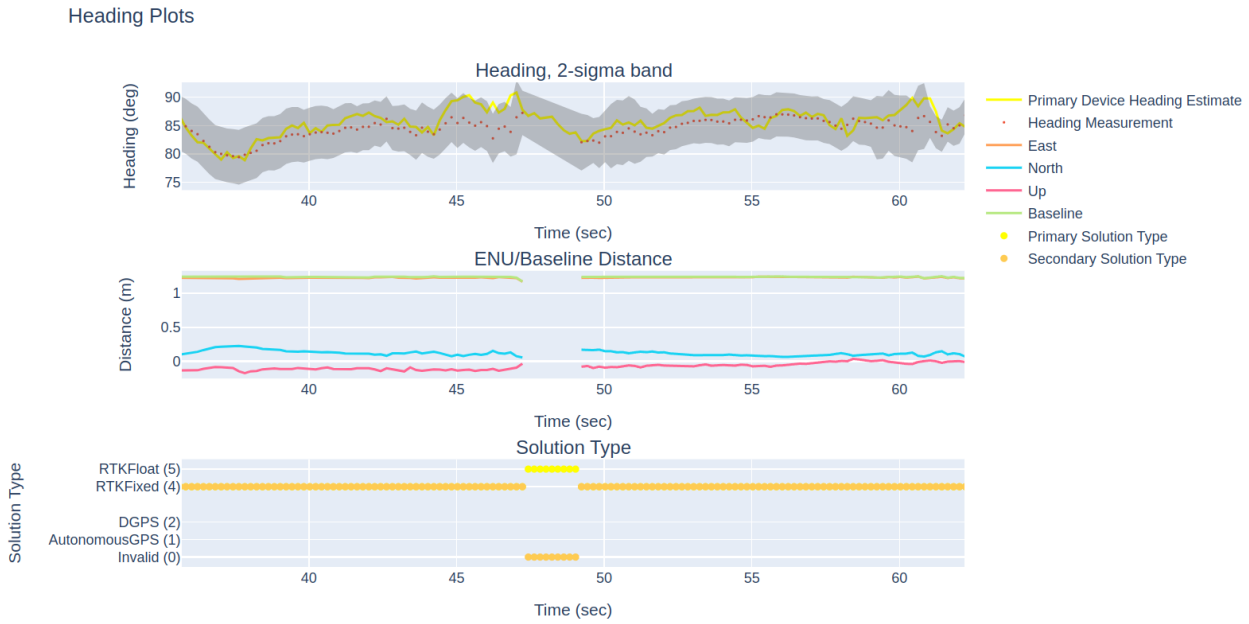
Alternatively, you can record a log by using the `p1_runner` tool included in the firmware release. The details of running this script are available in Section 4.2.

Once you have recorded a log file, you can generate plots using the following command:

```
p1_display <log_guid>
```

Where the `<log_guid>` field represents the first 4 or more letters/numbers of the log folder. When capturing a log in Point One Desktop, this ID will be displayed in the tray or in the table on the Logs page. You can also supply the entire path to the log file or its parent directory if you prefer.

This application will generate a series of HTML based plots. The *Measurements: Heading* plot displays the output from the secondary device's heading measurements and additional details:



Appendix A. Version History

Note: This section lists changes to this document only. For a list of changes for AH firmware updates, see the Point One Standard Dev User Manual.

Version 1.2 (2023-6-16)

- Set default configuration to be connecting both devices via UART2

Version 1.2 (2023-5-1)

- Added 4.2 for validation using p1_runner
- Clarified that RTK corrections should only be sent to the primary (AM/AP) device, not the secondary (AH) device
- 10 Hz output now supported as of AH 0.3.2 release

Version 1.1 (2023-3-13)

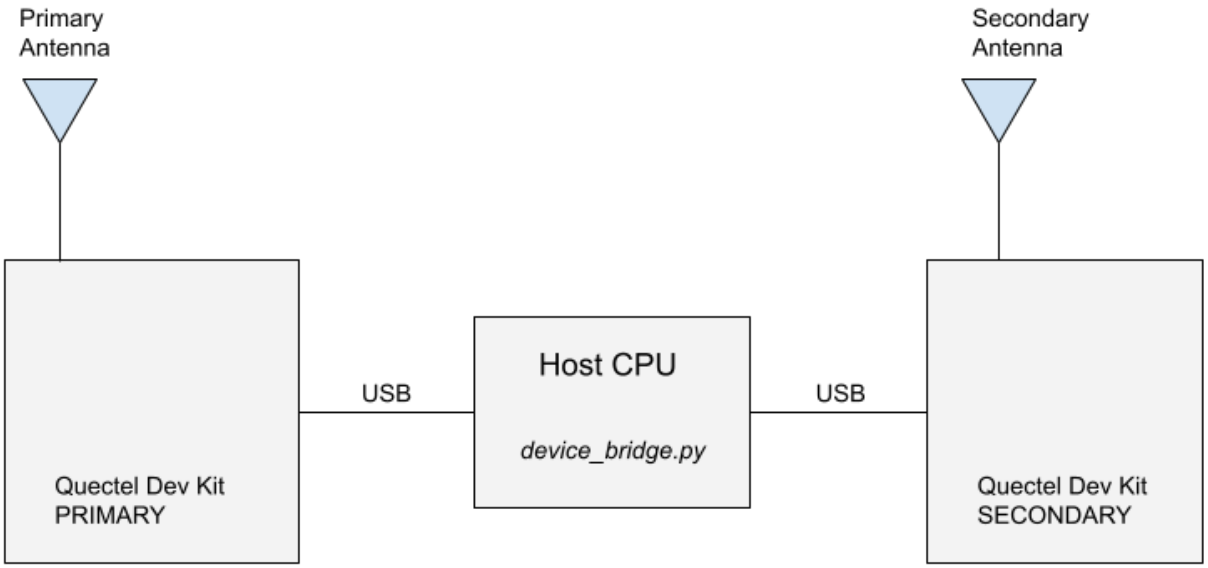
- Added Appendix D, for operation with different UART orientations

Version 1.0 (2023-2-17)

- Initial version of this document, corresponding with the AH 0.1.0 release.

Appendix B. Configuring the Quectel EVK as a Heading Device

B.1 Using the Quectel EVK with *device_bridge*

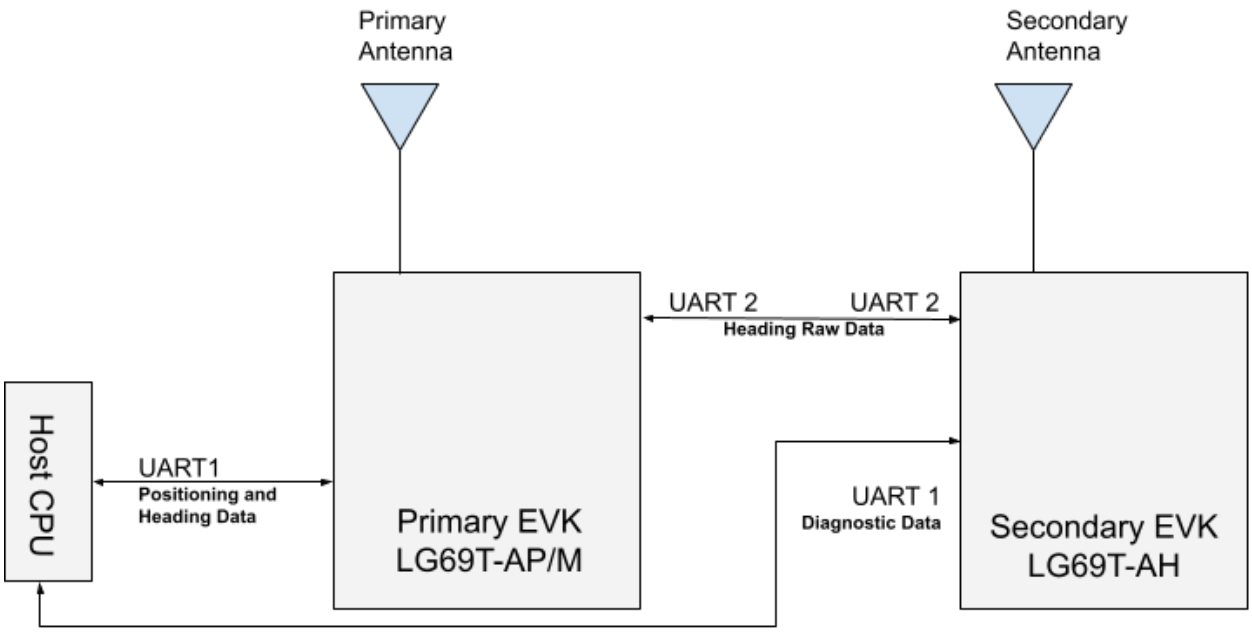


The Quectel EVK can also be used with the `device_bridge.py` software and wired similarly to the Point One Standard Dev Kit. However, note that the mapping of the UARTs to the Standard/Enhanced port names and the tty numbers is opposite that of the Standard Dev Kit.

For the Quectel EVK Only:

	Windows	**Linux**	**Mac**
UART1	Enhanced COM Port	`/dev/ttyUSB0`	`/dev/tty.SLAB_USBtoUART`
UART2	Standard COM Port	`/dev/ttyUSB1`	`/dev/tty.SLAB_USBtoUART1`

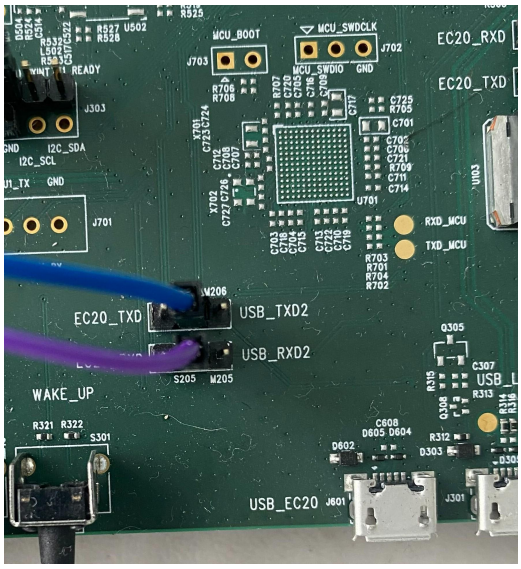
B.2 Using the Quectel EVK with Direct Module Wiring



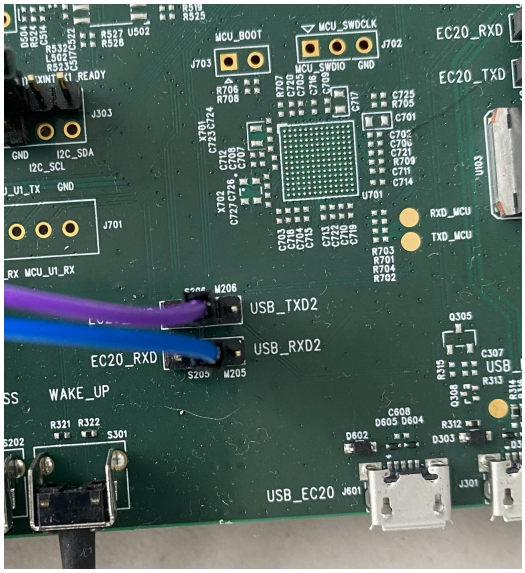
The Quectel EVK exposes the raw serial data from each module which allows a direct wiring configuration. This can also be realized in a customer design when directly using modules on a custom PCB. It is worth pointing out the signal levels are TTL between modules, so care should be taken to limit the length of these lines or to use a signal level converter (i.e. TTL/RS232).

To directly wire the EVKs, follow these steps:

1) On the Primary EVK, remove the jumpers on the USB_TXD2 and USB_RXD2 lines and replace them with jumper cables (not provided).



2) On the Secondary EVK, remove the jumpers on the USB_TXD2 and USB_RXD2 lines. Connect the USB_TXD2 from the Primary to the USB_RXD2 on the Secondary. Connect the USB_RXD2 from the Primary to the USB_TXD2 on the Secondary.



Note that this modification will disable host computer USB access to UART2 on the Primary and UART2 on the Secondary.

Now follow Section 4 to verify functionality.

Appendix C. Configuring udev Rules Under Linux

In Linux, it is possible to configure the `udev` subsystem to automatically detect your devices each time they are connected, and then assign consistent device names to them. For example, instead of `/dev/ttyUSB3`, which may change when you unplug the device, you can use udev to create a recognizable name such as `/dev/p1sdk-secondary-uart1` that will not change when the device is unplugged or the computer restarts.

1. Run `udevadm monitor` to monitor activity when a serial (TTY) device is plugged in. We'll use this to determine the path and serial number for each device.
   ```
   udevadm monitor -p -s tty
   ```
2. Unplug your primary Point One Standard Dev Kit (or Quectel EVK) devices and plug it back in.
3. You should see a series of prints. The last one will look similar to the following:
   ```
   UDEV  [4430.413678] add      /devices/.../ttyUSB3 (tty)
   …
   SUBSYSTEM=tty
   DEVNAME=/dev/ttyUSB3
   …
   ID_MODEL=CP2105_Dual_USB_to_UART_Bridge_Controller
   …
   ID_SERIAL_SHORT=00D38F50
   ```
4. Write down the serial number for the device, highlighted in bold above.
5. In a text editor, create a file named `/etc/udev/rules.d/99-p1sdk-primary.rules` with the following contents, replacing the highlighted serial number with the value noted above.

   Note that you will need to open the text editor with `sudo` in order to enable write permission for the udev rules directory.

   ```
   SUBSYSTEM=="tty", ATTRS{idVendor}=="10c4", ATTRS{idProduct}=="ea70",
   ATTRS{serial}=="00D38F50", GOTO="configure"
   GOTO="end"
   LABEL="configure"
   ATTRS{bInterfaceNumber}=="00", SYMLINK+="p1sdk-primary-uart2"
   ATTRS{bInterfaceNumber}=="01", SYMLINK+="p1sdk-primary-uart1"
   LABEL="end"
   ```
6. You can now refer to your device as `/dev/p1sdk-primary-uart1` (or uart2) for all tools, including `device_bridge.py` and `config_tool.py`.
7. Repeat these steps for your secondary device, replacing "primary" with "secondary" in the script where highlighted above.

Note that for a Quectel EVK, the interface and UART numbering is reversed as follows:

```
ATTRS{bInterfaceNumber}=="00", SYMLINK+="quectel-evk-primary-uart1"
ATTRS{bInterfaceNumber}=="01", SYMLINK+="quectel-evk-primary-uart2"
```

Appendix D. Changes from Recommended Configuration

This document details the process of connecting the devices via UART2 of the primary, to UART1 of the secondary device. Although this is the recommended set up, it is not required, but there are just a few considerations that need to be taken.

Regarding the primary device, the UART that is connected to the host computer either via Point One Desktop Application or p1_runner, must have both the Fusion Engine heading message and diagnostics enabled, using the command below:

```
./bin/config_tool.py --device-port /dev/(Primary UART to host computer) apply uart(1 or 2)_message_rate fe heading on -f
```

```
./bin/config_tool.py --device-port /dev/(Primary UART to host computer) apply uart(1 or 2)_diagnostics_enabled true
```

```
./bin/config_tool.py --device-port /dev/(Primary UART to host computer) save
```

Still looking at the primary device, the UART that is connected and transmits data to the secondary device must have diagnostic message output enabled. In order to do so, use the commands below:

```
./bin/config_tool.py --device-port /dev/(Primary UART to Secondary Device) apply uart(1 or 2)_diagnostics_enabled true
```

```
./bin/config_tool.py --device-port /dev/(Primary UART to Secondary Device) save
```

Turning our attention to the secondary device, the UART that is connected to the host computer either via Point One Desktop Application or p1_runner, must have both the Fusion Engine heading message and diagnostics enabled, using the command below:

```
./bin/config_tool.py --device-port /dev/(Secondary UART to host computer) apply uart(1 or 2)_message_rate fe heading on -f
```

```
./bin/config_tool.py --device-port /dev/(Secondary UART to host computer) apply uart(1 or 2)_diagnostics_enabled true
```

```
./bin/config_tool.py --device-port /dev/(Secondary UART to host computer) save
```

Still looking at the secondary device, the UART that is connected and receives data from the primary device must have diagnostics *disabled*. In order to do so, use the commands below:

```
./bin/config_tool.py --device-port /dev/(Secondary UART from Primary Device) apply uart(1 or 2)_diagnostics_enabled false
```

```
./bin/config_tool.py --device-port /dev/(Secondary UART from Primary Device) save
```