Notes about UART and baudrates

Proxmark3 RDV4 can interact with the client program running on a host via several means, let's go through each of them and study the notion of baudrate.

Table of Contents

USB-CDC ACM

^Top

USB CDC (Communications Device Class) ACM (Abstract Control Model) is a way to emulate serial ports over USB. On the host, it

appears as a virtual serial port, e.g. `/dev/ttyACM0`.

This is the basic way to communicate with all versions of Proxmark3 (with fairly recent FW, it used to be USB-HID-based long time ago). On the Proxmark side, USB-CDC is provided by `common/usb_cdc.c`.

In the USB CDC ACM descriptors, the Proxmark3 advertises a baudrate (USART_BAUD_RATE) but this is purely informative and has no real meaning. The real communication speed relates only to the USB link and is roughly about 7Mbps. In USB CDC ACM the host could "set" other baudrates and the USB CDC device would be informed of the changes (see `SET_LINE_CODING` and `GET_LINE_CODING`), which is potentially interesting if, behind, it has to configure a real UART, but we're not in this type of setup so baudrate notion on USB CDC (visible e.g. with `stty -F /dev/ttyACM0`) can simply be totally ignored.

Proxmark3 FPC USART

^Top

Proxmark3 RDV4 has a FPC connector outputting on 2 pins a USART from the ARM:

FPC	ARM	Function	GPIO
6	11	RXD1	PA21
7	14	TXD1	PA22

USART support is in `common/usart.c`.

There are mainly two ways to use this USART:

- connect the host client to the Proxmark3 via this USART instead of

USB-CDC, this is the `FPC_USART_HOST` option you can add to `PLATFORM_EXTRAS` in `Makefile.platform`. The most used way is through the BT add-on (blue shark) that we will cover later. Instead of BT add-on, we can also use e.g. a FTDI cable (mostly for internal development, it's much slower than USB-CDC anyway) or in the future other ways to connect the host such as a USART-to-Wi-Fi bridge.
- connect "slave" devices to the Proxmark3 to add functionalities. In such case, the host client will use USB-CDC and the USART will be use to, e.g. connect the Proxmark3 to various daughterboards. These is no such example of daughterboard as of today, except when we're talking to the BT add-on in its AT configuration mode.

This USART can be reached from the host client (if connected via USB-CDC) through the following commands, available when you add `FPC_USART_DEV` to `PLATFORM_EXTRAS` in `Makefile.platform`:

- `usart config`, to configure the baudrate and the parity of the Proxmark3 USART
- `usart txrx/tx/rx/txhex/rxhex` to transmit and receive bytes

So, `usart config` changes the Proxmark3 USART baudrate and parity, while e.g. `usart txrx -d "AT+Px"` and `usart txrx -d "AT+BAUDx"` changes the BT add-on parity and baudrate. And for things to work fine, both sets have to match!

Internally, the desired baudrate is converted to UART settings: a BRGR and a FP. The resulting baudrate will be close to but not always equal to the desired baudrate. Serial ports typically have some error tolerance in the actual baudrates. Theoretically < 2.5% on each side (so 5% in

total), < 2% to be on the safe side. In the current firmware configuration, the Proxmark3 can provide any baudrate up to 2Mbauds with an error of max 2%, and selected baudrates up to 6Mbauds (tested with a FTDI C232HM DDHSL-0 cable).

Proxmark3 FPC USART + BT add-on (blue shark)

^Top

BT add-on AT configuration mode

^Top

When the BT add-on is turned on but no actively connected to a host, it's in a configuration mode where it accepts "AT" commands and its blue LED is blinking at about 1Hz.

Some specific commands are available when you add `BTADDON` to `PLATFORM_EXTRAS` in `Makefile.platform` (it will automatically enable `FPC_USART_HOST` as well), to configure specific features of the BT add-on:

- `usart btpin`, to change the BT add-on PIN
- `usart btfactory`, to guess the current BT add-on UART settings and to reset its configuration.

`usart btfactory` changes several times the Proxmark3 USART baudrate and parity till it matches the BT add-on settings, then changes the baudrate and parity of the add-on to a default value, then changes the Proxmark USART to the same default values, so

everything should be back in order. (`btfactory` does more but we're only interested in baudrate in this discussion)

Manual configuration is also possible with `usart txrx -d "AT+Px"` and `usart txrx -d "AT+BAUDx"`.

BT add-on connected mode

^Top

When the BT add-on is paired with a host and the host establishes an active connection, the blue LED turns on steadily.

The add-on acts as a bridge, between its UART and the BT communication channel, here a RFCOMM channel.

RFCOMM

^Top

The Bluetooth RFCOMM protocol provides an emulation of serial ports over the L2CAP protocol (ref).

As for USB-CDC, the real speed of the link is unrelated to serial baudrate notion. Literature mentions a maximal value of 360kbps for some implementations, but the HC-06 Bluetooth module within the BT add-on is limited as the vast majority of similar devices to 128kbps.

BT add-on baudrate

^Top

Which baudrate will make sense?

- The Proxmark can set its USART baudrate to any value with a decent error up to large values.
- The BT add-on supports a set of fixed baudrates: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 1382400
- The HC-06 RFCOMM is limited to 128kbps (and has an internal 8Mbit buffer)

Using 115200 is safe and within the 128kbps limit.

Using 230400 allows to maximize the BT channel capacity, but the gain is limited, about 10-15% faster than 115200. There is also a risk to take into account: you're delivering data too fast to the HC-06 than what it's capable to send over RFCOMM. So you're filling the internal buffer faster than it can be emptied. If you're doing it for too many data, you'll reach a point where data will be lost (once the internal buffer is full), which is observable with `hw status` for higher baudrates at the time of writing (`b17da830edadb8462e02a95a00b4a58302cce71b`).

BT on host side: internal BT

^Top

On the other side of the BT link is the host. If it has built-in Bluetooth, the host can present a virtual serial port to the Proxmark3 client. E.g. on Linux, `rfcomm` allows to create such bindings of a BT device and a virtual port: `rfcomm bind rfcomm0 20:19:04:20:04:08` will create the virtual port `/dev/rfcomm0`. As any virtual serial port you may set a specific baudrate to that port, but it is meaningless as anyway the

RFCOMM link will run at 128kbps (at least when talking to HC-06 add-on).

Note that the rfcomm implementation separates pairing from actual connection:

- you will pair once at the very first connection, pairing will be remembered for next connections (pairing will be lost if you reconfigure the BT add-on settings).
- every time `/dev/rfcomm0` is opened by a program, e.g. Proxmark3 client, it will establish the link and the BT add-on blue LED will be on steadily.
- when the program closes, the link will be closed and the BT add-on blue LED will blink again, showing that the BT ad-on is in AT configuration mode.

BT on host side: HC-06 dongle

^Top

On the other side of the BT link is the host. If it does not have a built-in Bluetooth, the host can use

- a generic BT add-on, and we're back in the previous case
- or a BT-UART bridge as the one sold with the BT add-on, which is what we'll discuss in this section

It's actually bundling a HC-06 and some USB-UART chip together in a single dongle.

The HC-06 is the same as the one in the BT add-on, but working in *master* mode. When it's configured with the same settings as the BT

add-on, it will connect automatically to it. One difference with native BT seen in the previous section is that the link will be always active (BT add-on blue LED and dongle HC-06 LED will both be steadily on). So if you need to enter AT configuration mode on the BT add-on or on the dongle, you need to turn off the other device to break the RFCOMM connection.

When the dongle is plugged on the host, a serial port is allocated, `/dev/ttyUSB0` on Linux. Contrarily to the `/dev/ttyACM0` of the Proxmark3, `/dev/ttyUSB0` is controlling a real UART, the one in the USB-UART chip of the dongle, probably a CP2102.

For the host to talk properly to the HC-06 dongle, the USB-UART chip has to use the exact same baudrate as the one configured into the HC-06 dongle.

The USB-UART speed is selected when a program opens the port. E.g.

- the Proxmark3 client:
  - when launched, via the option `–b`
  - when reconnecting, via option `b` of `hw connect`
- the Python script meant to configure HC-06 dongle. The baudrate is given as option to `serial.Serial()` when opening `/dev/ttyUSB0`.

HC-06 dongle AT configuration mode

^Top

When the BT add-on is turned off, the HC-06 dongle will not be connected and will fall back into its AT configuration mode (its LED is

blinking).

`tools/btaddon/hc06_factory.py` changes several times the USB-UART baudrate and parity till it matches the HC-06 dongle settings, then changes the baudrate and parity of the dongle to a default value. (`hc06_factory.py` does more but we're only interested in baudrate in this discussion)

Proxmark3 FPC USART + FTDI

^Top

Just for completeness, let's mention the possibility to connect the Proxmark3 to the host via FPC and a USB-UART bridge such as a FTDI cable (or CP2102, ch341, PL2303...). It's not very interesting because USB-CDC is faster anyway.

In that case, Proxmark3 USART settings (configured e.g. via `usart config` while connected by USB-CDC) have to match FTDI cable settings, selected by the program opening the serial port (`/dev/ttyUSB0`).

BT add-on + FTDI

^Top

Just for completeness, let's mention the possibility to connect the BT add-on to the host via FPC and a USB-UART bridge such as a FTDI cable (or CP2102, ch341, PL2303...). The only interest is for debug purposes, e.g. to reconfigure an add-on from the host rather than from the Proxmark3 itself. It was used at some point to recover manually

from BAUDB/BAUDC but now the PRoxmark3 firmware supports directly these high baudrates.

Connect a FPC breakout to the BT add-on. The FPC breakout is meant to be connected to the Proxmark3, not the add-on, so the pinout order is reversed. The RX/TX logic is also reversed: TX referred to Proxmark3 TX, so now we must connect FTDI TX to TX and not RX. Add-on gets powered externally via the "battery" pin.

FTDI	FPC breakout	Add-on
GND	(PWR_OE1)	GND
CTS		
5V	(LF_VMID)	Vin?
TXO	(5V)	RX
RXI	(HF_IN)	TX
DTR		

Turn BTpower switch ON, leave Battery switch OFF

Use e.g. `tools/btaddon/hc06_factory.py` with `role = b'S'`

HC-06 AT Commands

^Top

The HC-06 supports a subset of the AT commands and they are listed below. The commands are limited and vary in how to give parameters. A pure informative command like AT+VERSION just returns a string with current firmware the HC-06 has installed.

The BAUD/PIN/NAME/P/LED commands take their parameter directly

after the command. E.g.:

```
AT+BAUD8
       ^   parameter
   ^^^^    command
```

The "8" indicates a baudrate of 115200.

The ROLE command takes its parameter after an equal sign:

```
AT+ROLE=S
       ^  parameter
   ^^^^    command
```

Command	Description	Reply
`AT+VERSION`	print firmware version	`hc01.comV2.0`
`AT+BAUD8`	set baudrate to 115200	`OK115200`
`AT+PIN1234`	set PINCODE to 1234. (must be numbers)	`OKsetPIN`
`AT+NAMEPM3_RDV4.0`	set device name (as shown to BT hosts) to `PM3_RDV4.0`	`OKsetname`
`AT+ROLE=S`	set device role to *slave*	`OK+ROLE:S`
`AT+ROLE=M`	set device role to *master*	`OK+ROLE:M`
`AT+LED0`	turn LED off	`LED OFF`
`AT+LED1`	turn LED on	`LED ON`
`AT+PN`	set NO parity	`OK None`
`AT+PE`	set EVEN parity	`OK Even`
`AT+PO`	set ODD parity	`OK Odd`

We recommend you to use the scripts when trying to set these

parameters.

We've noticed that the HC-06 in Master mode doesn't always reply properly to the commands, especially to AT+VERSION. If you have such a problem, temporarily switch it to Slave mode, power cycle it and send again AT+VERSION.