Mercury System

MS_AN_002

Use Mercury Serial Terminal

Francesco Ficili
17/09/2017

Revision Log				
Author	**Date**	**Major**	**Minor**	**Description**
Francesco Ficili	17/09/2017	1	0	Initial version.

Table of Content

1. Introduction

This manual describe in details how to use the serial terminal integrated in the Mercury System framework. The terminal could be accessed by one of the following interfaces of the Base Boards:

- USB device port,

Initially the terminal defaults to the USB port, enabling a serial emulation over USB protocol, so from the remote OS point of view, it could be accessed using the legacy serial interface.

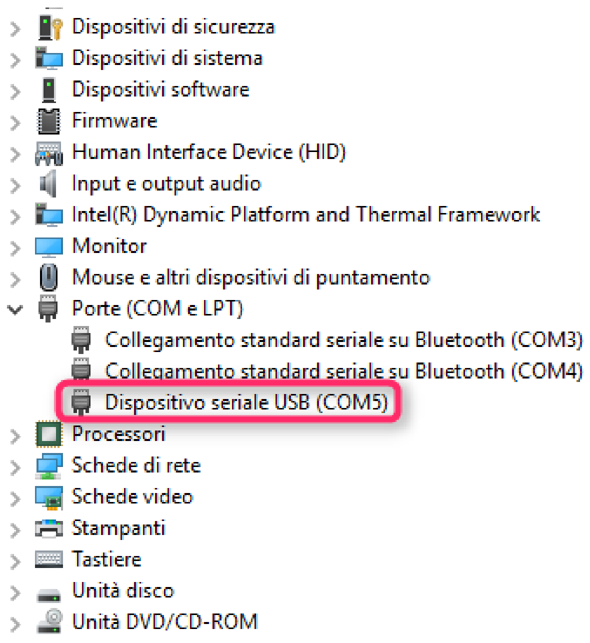
2. USB Drivers

Depending on the OS of you system you could be requested to install a driver, in order to use the USB.

Windows

Most recent versions of Windows (Win 8/10) are able to automatically install a proper driver without user intervention. If you use Win7 or XP, the system will ask for a .inf description file you need to point to.

Once the driver is installed you need to llok at the device manager to understand which COM port the device is associated to:



```
> Dispositivi di sicurezza
> Dispositivi di sistema
> Dispositivi software
> Firmware
> Human Interface Device (HID)
> Input e output audio
> Intel(R) Dynamic Platform and Thermal Framework
> Monitor
> Mouse e altri dispositivi di puntamento
v Porte (COM e LPT)
      Collegamento standard seriale su Bluetooth (COM3)
      Collegamento standard seriale su Bluetooth (COM4)
      Dispositivo seriale USB (COM5)
> Processori
> Schede di rete
> Schede video
> Stampanti
> Tastiere
> Unità disco
> Unità DVD/CD-ROM
```

Linux

Upon plugging in a USB CDC ACM virtual COM port device into a Linux machine, the OS will automatically enumerate the USB device successfully, and a new object should show up as:

```
/dev/ttyACMx
```

(where ttyACMx is usually ttyACM0, but could be some other number such as ttyACM1, if some other ACM device is already attached to the machine). To determine the exact number value of "x", a procedure like follows can be used:

- Open a console.
- Make sure the USB device has been plugged into the machine.
- Type: lsusb
- You should see a line like: Bus 005 Device 004: ID 04d8:000a Microchip Technology, Inc.
- Type: modprobe cdc-acm vendor=0x04d8 product=0x000a
- Type: dmesg
- You should get the status, showing the ttyACMx value, ex: cdc_acm 5-1:1.0: ttyACM0: USB ACM device

MAC Os

Upon plugging in a USB CDC ACM virtual COM port device into a Mac OS X based machine, the OS should automatically enumerate the USB device successfully, and a new object should show up as:

```
/dev/tty.usbmodemXXXX
```

(where XXXX is some value, such as “3d11”)

To run the example demo project: “USB\Device - CDC - Basic Demo” on a Mac OS X based machine, a procedure like follows can be used:

Open TERMINAL. This can be done by clicking SPOTLIGHT and searching for TERMINAL. Spotlight is the little magnifying glass in the upper right of the screen. In Terminal, with the USB CDC ACM device NOT plugged in (yet), type:

```
ls /dev/tty.*
```

This will show all serial devices currently connected to the Mac. In the author’s case, the following list appears:

```
/dev/tty.Bluetooth-Modem

/dev/tty.Bluetooth-PDA-Sync

/dev/tty.Rob-1
```

Now, plug the USB CDC device into a USB port of the Mac. Hit the UP cursor, which will bring the search command back (ls /dev/tty.*) and hit return. You should get the exact same list as before, but this time, with a new serial device. In the author’s case, it was:

```
/dev/tty.usbmodem3d11
```

Once the complete name is know, the received serial port data can be displayed by typing:

```
screen /dev/tty.usbmodem3d11
```

(replace “3d11” in the above line with the value for your machine). If the microcontroller was programmed with the “USB\Device - CDC - Basic Demo”, you can then press the user pushbutton, and the standard demo text should be printed to the screen (ex: “BUTTON PRESSED ---”).

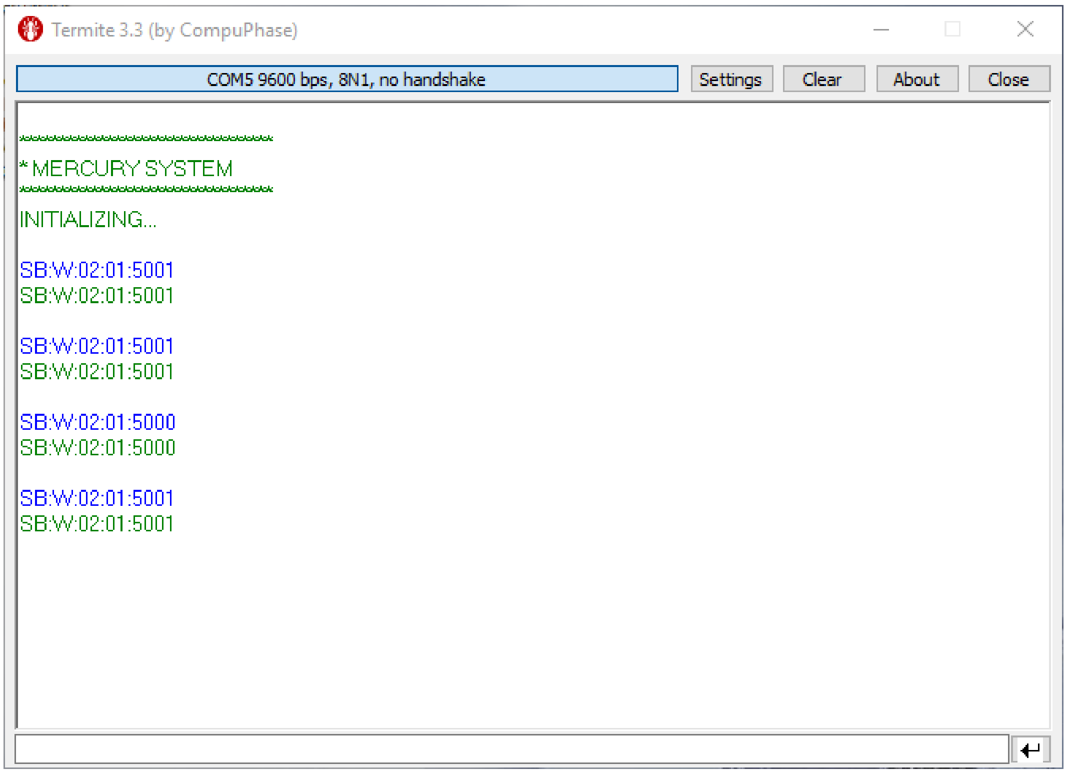
If the USB device is being operated as a USB to UART translator device (ex: using “USB\Device - CDC - Serial Emulator” firmware, the baud rate can be set by using syntax like follows:

```
screen -U /dev/tty.usbmodem3d11 38400
```

Where “usbmodem3d11” should be replaced with the actual value of the device, and “38400” should be replaced with actual desired baud rate (ex: 9600, 19200, 38400, 57600, 115200, etc.). More details and usage information for screen can be found in the man page.

3. Remote PC Software

Any terminal program able to interact with the system serial port could be used. A good program for Windows is Termite, by CompuPhase:



```
*****************************
* MERCURY SYSTEM
*****************************
INITIALIZING...

SB:W:02:01:5001
SB:W:02:01:5001

SB:W:02:01:5001
SB:W:02:01:5001

SB:W:02:01:5000
SB:W:02:01:5000

SB:W:02:01:5001
SB:W:02:01:5001
```

4. The Mercury Terminal Protocol

Once the communication is active, you can use the terminal commands. These are the function currently supported by the terminal:

- Adjust terminal options,
- Communication with I2C slave micro-network

The terminal implements the following message format:

[op_type]:[rw]:[len]:[addr]:[payload]

Where:

[op_type]: is the operation type, and could be:

- SO: System Option
- SB: Slave Board control.

[rw]: is the operation that you want to perform, and could be:

- R: read operation,
- W: write operation.

[len]: length of the payload (2 byte ASCII base 10).

[addr]: is the address of the slave device you want to talk to (2 byte ASCII base 10).

[payload]: communication packet to the slave device, max 20 byte ASCII base 16.

Note: note that len and addr parameters use BASE 10, while payload uses BASE 16 for a more comfortable usage of the terminal.

5. Examples

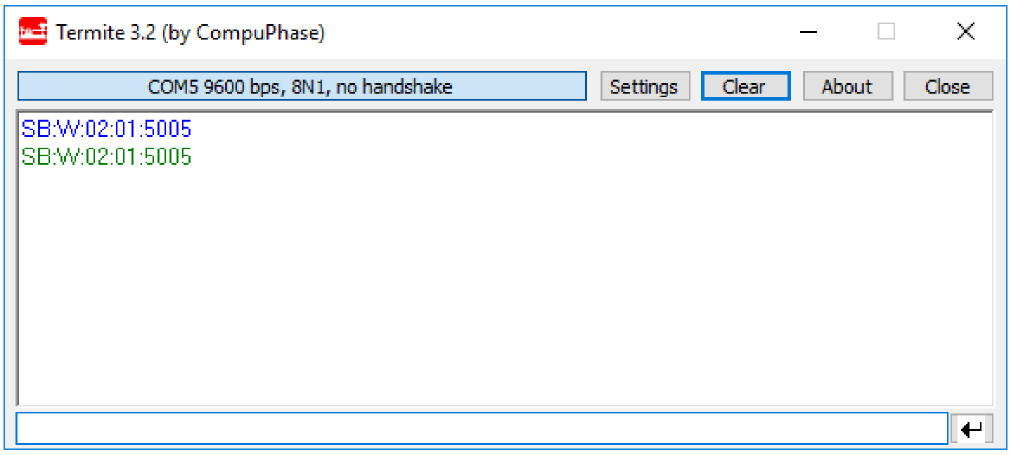
This chapter present some usage examples of the Mercury Terminal. The terminal feature allows to implement complete application without even change the FW of the embedded side, exploiting the high level functionalities implemented inside the slaves.

SB140 HSD Controls

The SB140 is a HSB bank module, which allows to control up to 4 loads up to 500mA per load. The following example shows how to control the HSD output status. The command used to control the output status for SB140 is 0x50 followed by 1 parameter which lists the status of the 4 output lines as bitfield (so 0x00 → all OFF, 0x01 → line 1 ON, other off, 0x0F → all ON).

The following example shows how to switch line 1 and 3, with a SB140 with address 0x01:

[op_type]	‘SB’
[rw]	‘W’
[len]	‘02’
[addr]	‘01’
[payload]	‘5005’
Resulting command	SB:W:02:01:5005



```
SB:W:02:01:5005
SB:W:02:01:5005
```

Each command sent to the device is echoed back by default.

SB identification Service

By default, each slave device implement some basic services; one of them is the identification service, to which the slave replies with its model name (service 0x20).

The board interrogation comprises of two parts, the first one which is a write in which the host asks for the preparation required data, and the second one which is a read in which the host asks for the transmission of previously prepared data.

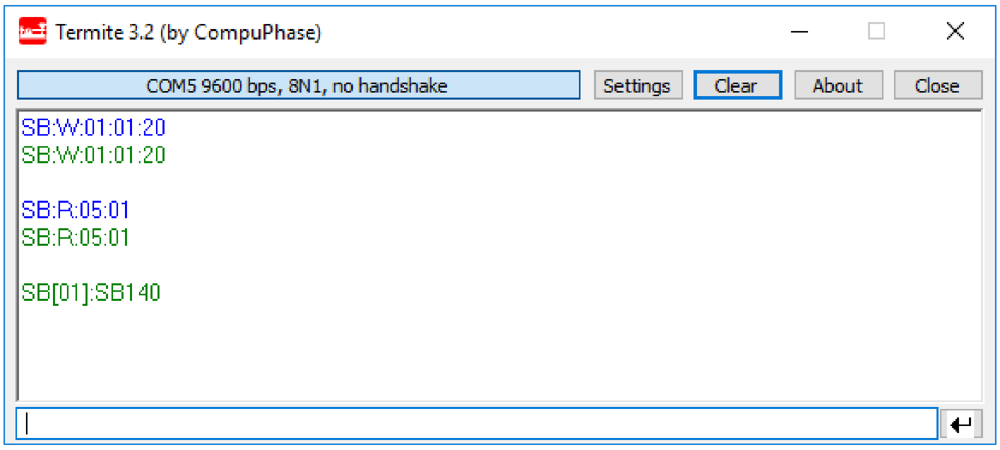
This example shows how to make this 2 step transaction:

Step 1: Data preparation request.

[op_type]	'SB'
[rw]	'W'
[len]	'01'
[addr]	'01'
[payload]	'20'
Resulting command	SB:W:01:01:20

Step 2: Data tramsission request.

[op_type]	'SB'
[rw]	'R'
[len]	'05'
[addr]	'01'
[payload]	Empty
Resulting command	SB:R:05:01



```
SB:W:01:01:20
SB:W:01:01:20

SB:R:05:01
SB:R:05:01

SB[01]:SB140
```