Panasonic

eUniStone

PAN1322 Application Note DesignGuide

Revision History

Revision Number	Description	Revision Date
1.2	Updated reference schematic Added chapter describing how to change from HCI to SPP, chapter 9.4.	16-Oct-2013
1.1	Added chapters Application development and FAQ	27-Jun-2013
1.0	New document for eUniStone PBA31309 v1.00	12-Mar-2013

Contents

Figures

Tables

1 Introduction

eUniStone (embedded UniStone), with part number PBA31309, is a module based on PMB8754 (eBMU), an integrated BT radio transceiver, baseband and protocol stack, with EEPROM, band-pass filter and built in antenna

eUniStone supports the following features

- Bluetooth v2.1 + EDR compliant SPP implementation
  - Secure Simple Pairing, Security Mode 4
    - Association Models „Numeric Comparison“, „Just Works“ and „Passkey Entry“ are supported
  - Encryption Pause Resume
  - Enhanced Power Control (BT3.0 feature of the BT Controller)
- Device A (initiating) and device B (accepting) role
- One point-to-point link for data transmission
  - octet by octet in stream mode
  - by packets in command mode (MTU size 500 bytes)
- Device is visible and connectable until the link has been set up
- Sniff and Sniff Sub Rating are supported on the link to save current
- Up to five trusted devices can be stored in EEPROM
  - when 6th device is paired, the first device is deleted
- AT commands for development and manufacturing
  - Device Under Test mode for connection to a BT tester
  - Secure Simple Paring Debug mode to sniff and decrypt the air traffic
  - Crystal oscillator re-calibration (needed for debug only)
  - EEPROM configuration update
  - SW upgrade via UART and I2C

- UART with HW flow control (RTS/CTS)
  - Use of HW flow control is mandatory
  - UART baud rate may be changed in EEPROM configuration 9.6kbps to 3.25Mbps

The block diagram of the eUniStone module is shown below.

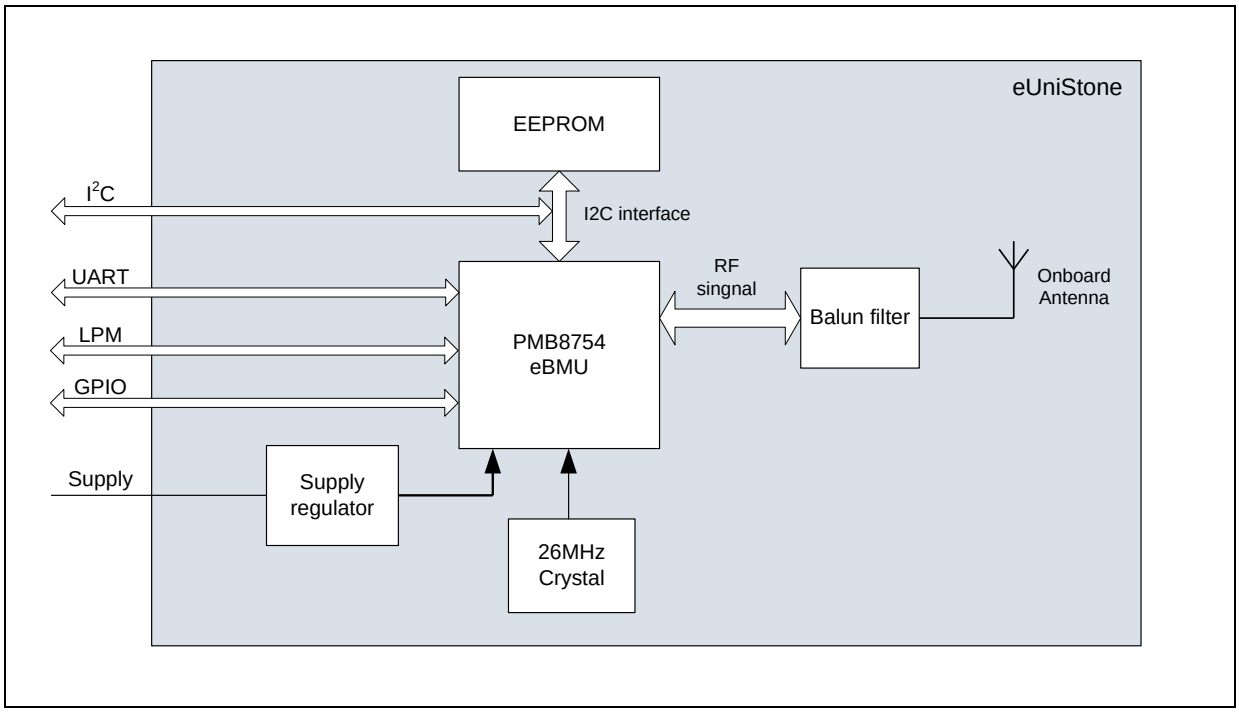


Figure 1. Simplified Block Diagram of eUniStone Module

2 Interfaces

2.1 UART Interface

The UART interface is the main communication interface between the host and eUniStone. For the SPP application, communication between the host and the eUniStone is through AT commands over the UART interface.

The interface consists of four UART signals for the AT interface and two GPIOs for additional low power mode control, as shown in Figure 2.

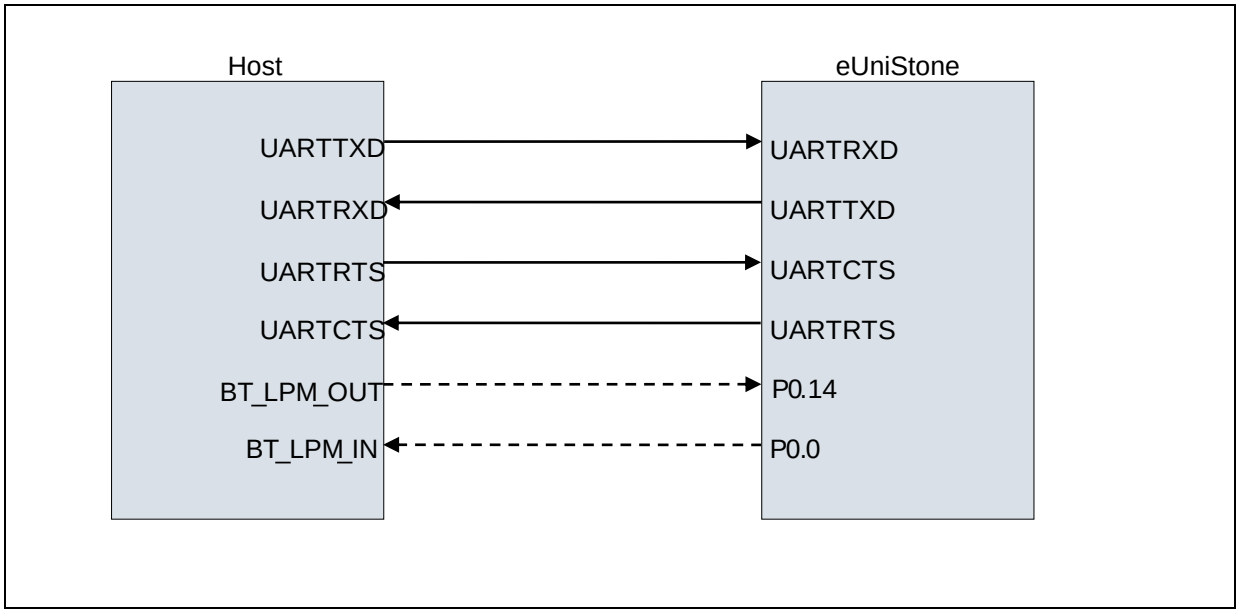


Figure 2. UART Interface

2.1.1 Low Power Mode Control

The use of low power mode (LPM) control is optional. If not used, P0.14 shall be tied to VDDUART.

The low power mode protocol for eUniStone is based on hardware signaling only. No AT commands or responses are required for the low power mode protocol. The two GPIOs are used to tell the other device (host or controller) when it may enter low power mode, when it should wake up and when it cannot transmit because the other device is in low power mode.

To allow the eUniStone to enter low power mode, the host sets pin P0.14 low. When eUniStone is ready, it will also allow the host to enter LPM by setting P0.0 low. Before entering LPM, the host shall set UART CTS of eUniStone high. Before entering LPM, eUniStone will set its own UART RTS high.

The host can wake up eUniStone by setting UARTCTS of eUniStone low again and setting P0.14 high again, whereas the eUniStone can wake up the host by setting its own UART RTS low again and setting P0.0 high again.

Figure 3 shows a case where the host initiates low power mode entry and exit.

Host		eUniStone
Host Output	→	P 0.14
Host RTS	→	UARTCTS
Host Input	←	P 0.00
Host CTS	←	UARTRTS

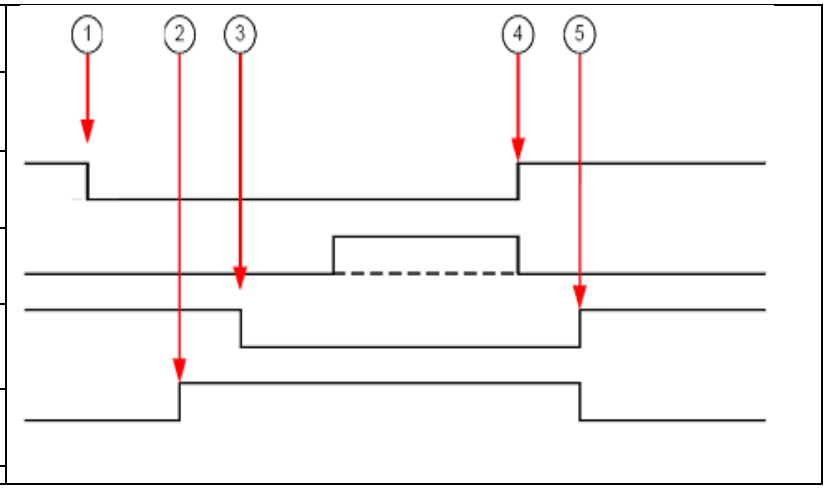


Figure 3. Host Initiates Low Power Mode Entry and Exit

1. The host allows eUniStone to enter low power mode
2. eUniStone starts to enter low power mode
3. When UARTRTS is high and P0.0 is low, eUniStone enters low power mode. The host may now enter low power mode by signaling UARTCTS high.
4. The host requests the eUniStone to wake up
5. eUniStone wakes up. P0.0 goes high and UARTRTS goes low.

Figure 4 shows a case where the host initiates low power mode and the controller wakes up the host.

Host		eUniStone
Host Output	→	P 0.14
Host RTS	→	UARTCTS
Host Input	←	P 0.0
Host CTS	←	UARTRTS

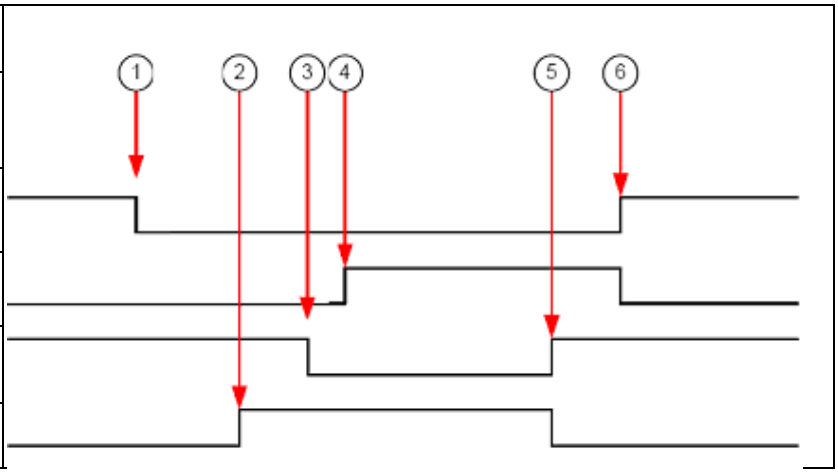


Figure 4. Host Initiates Low Power Mode Entry, eUniStone Initiates Exit

1. The host allows eUniStone to enter low power mode
2. eUniStone starts to enter low power mode
3. When UARTRTS is high and P0.0 is low, eUniStone enters low power mode. The host may now enter low power mode by signaling UARTCTS high.
4. The host enters low power mode.
5. eUniStone wakes up and requests the host to wake up
6. The host wakes up and accept UART data by lowering UARTCTS

2.1.2 Hardware Flow Control

Use of the UART HW flow control pins UARTRTS and UARTCTS is strongly recommended.

If the HW flow control is not used, UARTCTS must be tied low (flow "go"). Floating UARTCTS will result in random stop of the UART flow.

Without the HW flow control, a buffer overflow can occur on the UART, more probably at higher baud rate but also possible in lower baud rates due to congestion over the air.

For the LPM protocol, UARTCTS shall be high during LPM. In some cases the module could send events before the host is completely awake. Therefore LPM shall not be used without HW flow control.

2.2 EEPROM / I2C Interface

As shown in the block diagram (Figure 1), the eUniStone module is equipped with an EEPROM for storage of the non-volatile information. The size of the EEPROM is 32 kbit.

The EEPROM contains the following partitions:

- The Bluetooth Device Data (BD_DATA) Storage
- The application software
- Application specific data
- Production default values

On the PBA31309 module, the communication between the eBMU PMB 8754 and the EEPROM is carried through a standard I2C bus. The I2C bus is also accessible from external pins, which is useful in the development or debug phase in case the EEPROM gets corrupted.

2.3 GPIO Interface

Most digital pins on eUniStone can be used as general purpose I/O's (GPIOs). The GPIO pins are grouped into two ports: P0 and P1. P0 has 16 pins (P0.0 - P0.15) and P1 has nine pins (P1.0 - P1.8).

The non-reserved pins on port P0 may be controlled with AT commands. The P1 pins are not controllable through AT commands. They are reserved for use by future applications of the chip.

2.4 JTAG Interface

The pins used for the JTAG interface (TDI, TDO, TMS, TCK and RTCK) can also be used as general purpose I/Os. The operative interface (JTAG or GPIO) on these pins can be selected through the mode selection pin JTAG#. When JTAG# is connected to low, the pins are used for JTAG interface. When JTAG# is connected to high, the pins serve as GPIO pins.

JTAG# has an internal pull-up. If the JTAG functionality is not needed, leave this pin open.

3 Power Supply

- Main supply voltages (VSUPPLY1 on pin A4, A5 and A6) are required in the range from 2.9V to 4.1V. All these supplies are internally connected. It is only necessary to supply one of them.
- VDDUART defines the reference level for UART interface. It can be
  - supplied externally. In this case voltages between 1.35 and 3.6 V are required.
  - connected to the internal regulator, voltage “Internal 2” on pin VREG (pin No. C1), if 2.5 V is enough for UART operation (of course, voltage levels of the digital signals on the UART interface depends from this supply voltage). Ultimately, this depends on the host.

    Note: The state of UART pins is not defined while VDDUART is not supplied. The host shall not drive any UART pin before the reference levels are stable.
- VDD1 defines the reference level for ports P0.0-P0.3. It can be
  - supplied externally. In this case voltages between 1.35 and 3.6 V are required.
  - connected to the internal regulator, voltage “Internal 2” on pin VREG (pin No. C1), if not used.

    Note: The state of pins P0.0-P0.3 is not defined while VDD1 is not supplied. The host shall not drive any of these pins before the reference levels are stable.

3.1 Power up Sequence

The eBMU accesses the EEPROM to load the BD-Data and the application data during startup. The EEPROM access starts around 22 ms after the RESET# pin is pulled high. Power dropouts during the start-up phase can cause EEPROM data corruption. The startup typically takes 100ms. If power falls below the EEPROM’s minimum supply level during the startup phase, a reset shall be applied immediately. This can be achieved by an external pull-down on RESET#. There is no internal pull resistor on RESET# on the module.

The sequence of the EEPROM accesses at start-up is shown in Figure 5, During the EEPROM access, UARTRTS stays high. When the EEPROM access has finished, the UARTRTS signal is pulled down and a startup response “ROK” is sent via UART to the host. After receiving the startup response, the host is informed that the eUniStone module is ready to work.

The range for the startup time is indicated for each application SW release in the Release Notes [3].

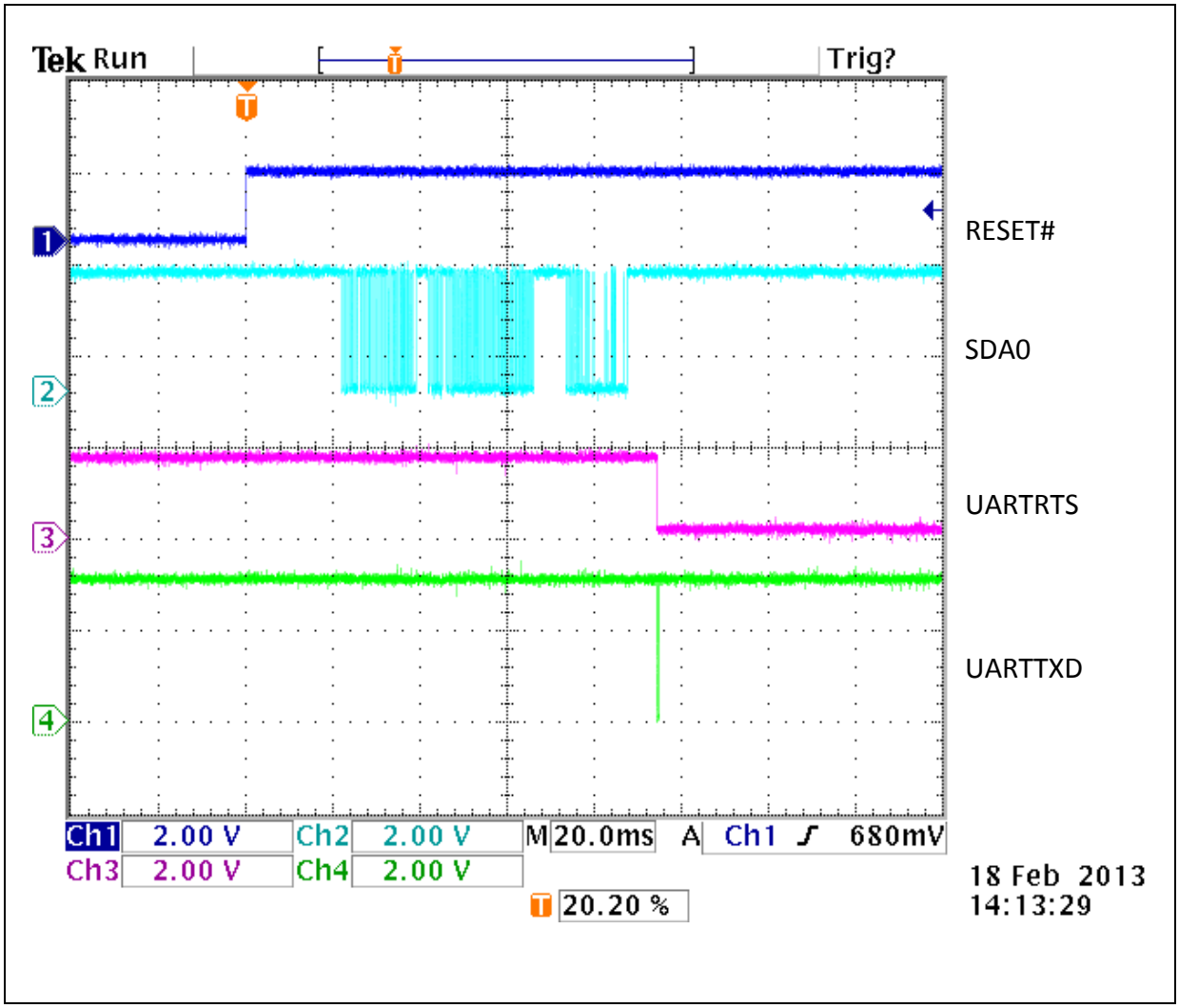


Figure 5. Example of EEPROM Access at Start-up

3.2 Power down through ONOFF Pin

If VSUPPLY, VDDUART and VDD1 are supplied by the same source, it is not possible to switch off the module with the ONOFF signal.

When the ONOFF pin is driven low, the reference levels VDDUART and VDD1 must also be switched off. Otherwise the module would draw current from VDDUART and VDD1, and the output pins would be “undefined”, i.e. they could drive high or low levels or vary periodically.

One option is to supply VDDUART and VDD1 by pin C1 VREG which connects to the “Internal2” voltage. Internal2 is switched off when ONOFF goes low. VSUPPLY can remain always on. Precondition for this is that the host’s UART pins are compatible with the logical levels for Internal2 driven pins indicated in the data sheet: Input Low max 0.45 V, Output High min 2.1 V. See “Table 18 Internal2 (2.5 V) Supplied Pins” in the data sheet [1] for the full specification.

If ONOFF is not used, it shall be connected to VSUPPLY.

4 Operation Modes

4.1 Command Mode

In command mode which is the default mode after power up or HW/SW reset, all communication by the host with the module is done using AT commands. The module will respond with different responses depending on what command was sent. Commands to eUniStone shall be ended with carriage return and line feed. Responses from eUniStone will be ended in the same way. E.g. for SW reset, AT+JRES<cr><lf> sent to eUniStone will be responded with ROK<cr><lf>. ("<cr><lf>" might be omitted in further descriptions of commands and responses)

After a SPP connection is set up to another device, data is sent using the command AT+JSDA. Data received from the other side is sent to the host with the response +RDAI. See example in the SPP AT specification, [2]. Sending and receiving data in command mode is normally used when transmitting burst and packetized data.

Table 1. Sending and receiving data in command mode

Direction	Command / Response	Note
Host → eUniStone	AT+JSDA=008,DataSent<cr><lf>	Data that shall be sent
Host ← eUniStone	OK<cr><lf>	Positive response that the data was sent
Host ← eUniStone	+RDAI=012,DataReceived<cr><lf>	Data received

4.2 Stream Mode

Stream mode can only be used after an established SPP connection. In this mode, the host application will send and receive un-packetized data to and from eUniStone, which are transmitted/received over the air to the remote device. Carriage return and line feed are not used in either direction, as in command mode. This mode is normally used when transmitting small size of data in a random way and for serial cable replacement applications.

Table 2. Sending and receiving data in stream mode

Direction	Stream data	Note
Host → eUniStone	DataSent	Data that shall be sent
Host ← eUniStone	DataReceived	Data received

4.3 Switch from Command Mode to Stream Mode

After a Bluetooth SPP connection has been established, the host can send the command AT+JSCR in command mode to switch the eUniStone to stream mode. A transparent communication will be enabled if both sides of the connection are using stream mode.

4.4 Switch from Stream Mode to Command Mode

The host can send the escape string "^ ^ ^" to switch the eUniStone back to command mode. The string must be sent with the following timing:

- Pause time between each "^ " sign: T1 = 100ms to 1 sec.
- Pause time after the last "^" sign: T0 > 1sec.

If the remote device disconnects the link when the eUniStone module is in stream mode, the eUniStone module will immediately send a „Disconnect Indication" +RDII to the host and switch back to command mode automatically.

If the remote device is off (e.g. powered off or out of range) when the eUniStone module is in stream mode, stream data cannot be transmitted any more on the air. Hardware flow control stops any further UART communication as soon as the module's UART input buffer is full. After the link supervision timeout of 20 s, the module sends a „Disconnect Indication" +RDII to the host. At that instant, Any UART data is flushed and the device switches back to command mode automatically.

The host has to wait during the supervision timeout of 20 s. As soon as the UART buffer is full, it is not possible to leave stream mode or to disconnect the link locally before the supervision timeout expires.

There are two ways to detect a disconnection in stream mode:

1. The host is searching for "+RDII" in the data stream from the module. The host shall switch from stream mode to command mode when it recognizes the "+RDII" indication. Any stream data sent after the "+RDII" indication will be answered by an error message "ERR=-1" (syntax error). If the host after that sends a first command it will be responded with "ERR=-1", since the UART is out of sync. The second command will get the expected response (e.g. "OK").
2. The GPIO pin P0.1 (pin E5) is used to indicate the connection status. P0.1 is HIGH when eUniStone device is connected and LOW when there is no connection. The transition from HIGH to LOW happens prior to sending the "+RDII" indication via UART. Hosts that cannot monitor the incoming data stream for the "+RDII" indication in stream mode may monitor P0.1 in eUniStone.

P0.1 is configured as input pin by default. To use this feature the host shall send the AT command "AT+JGPC=FFFD,0000,0000,0000,FFFD" which configures P0.1 as an output pin.

5 Accessory Development Compatible with Smart Phones

When using eUniStone in an accessory the design should contain eUniStone and a host processor that controls the module through the UART interface, see Figure 6. The host solution will need a host application. This application can run on a micro-controller, FPGA, PC or similar. When writing the host application, there are some things that need to be considered. A host solution with eUniStone that is communicating with mobile solutions on Android, iPhone and Windows phone 8 will need to take specific considerations for the host application depending on Smart Phone/OS. These items will be described in this chapter.

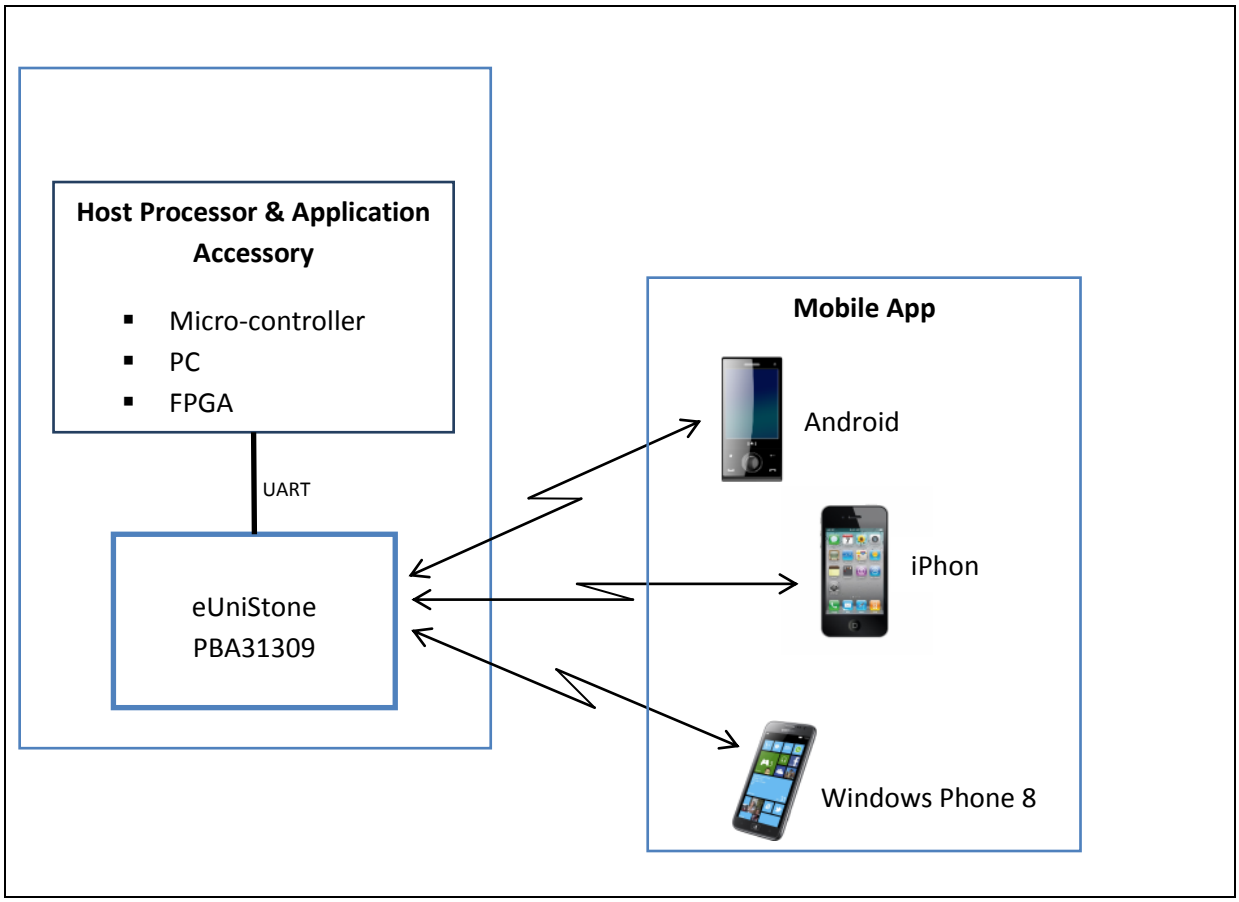


Figure 6. Accessory with Mobile App

- Accessory: eUniStone + Host processor + Host Application
- Mobile App: Mobile phone + Mobile application

The host application that controls eUniStone need to fulfill the control as required by the Accessory functionality. To control the Bluetooth link using eUniStone minimal requirements must be covered since the entire Bluetooth stack is included in eUniStone.

5.1 Accessory

An accessory can be any device that connects, sends and/or receives data to and from a remote device. The device can be small and portable or part of a bigger system. Typical usages for an accessory are;

- Sensor
- Display
- Remote control
- Display with buttons
- Keypad
- Handheld terminal

The eUniStone is specially made to support all these types of accessories/applications using the Serial Port Profile (SPP) over Bluetooth.

5.1.1 Host Processor

The host processor in an accessory, using eUniStone, may be a really light micro-controller, FPGA or similar but also e.g. a PC. In the below figure there is an external peripheral control and chip connected to the host processor, this may be needed depending on the accessory functionality requirements. For use in an iPhone application there is a need for an external chip and control functionality.

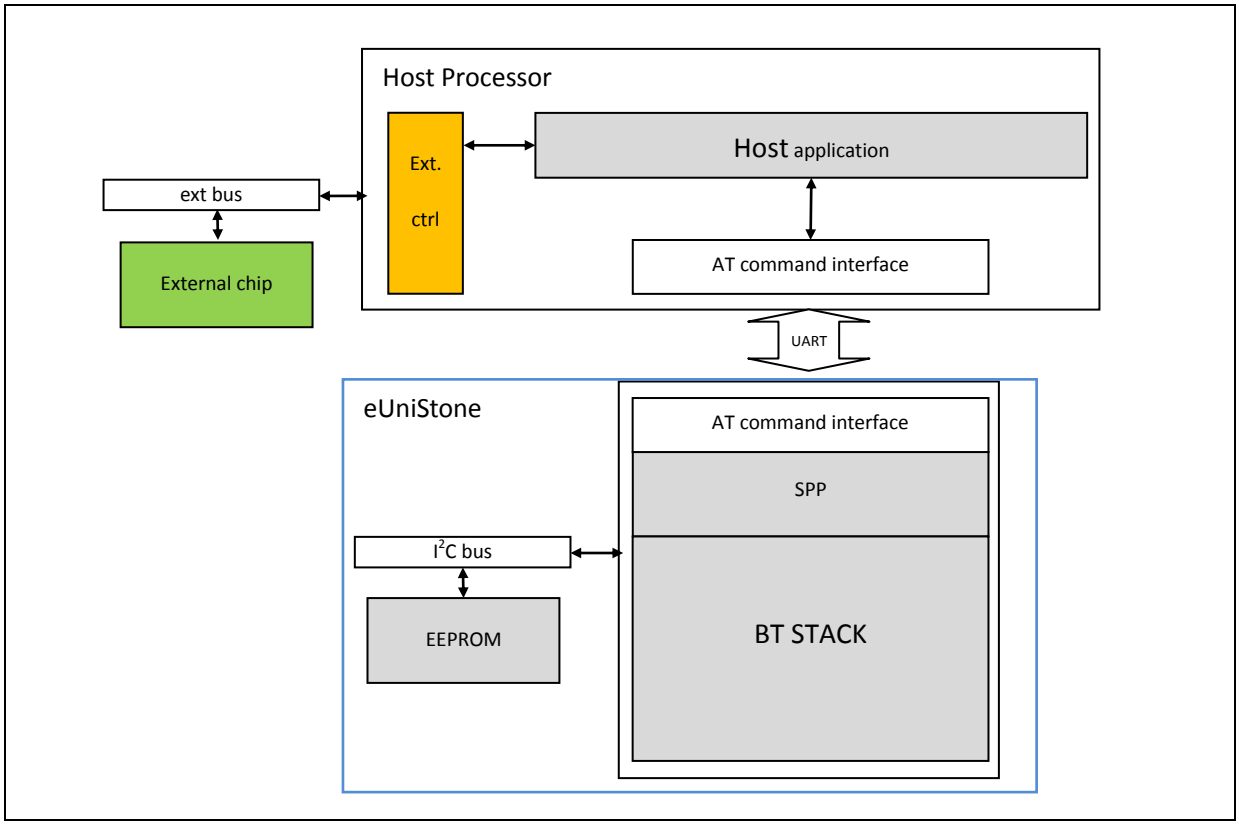


Figure 7. Host processor with peripheral device and UART connection to eUniStone

The processor is chosen depending on criteria like,

- Size, physical and memory
- Performance
- Power
- Other peripheral functions

Other peripheral function can be measurement of physical parameters like temperature, humidity, pressure, acceleration. Also input and output capabilities need to be considered like how keyboard, display, LED's and input buttons should be handled, and it all depends on the functionality of the complete accessory product.

The eUniStone provides the support to use a low cost minimal footprint host processor. A normal implementation for a host running an eUniStone could be around;

- Host application Flash: 4-8KB
- Application (RAM): 4-8KB
- Data buffering (RAM): 1-2KB

Of course the complete functionality decides the final requirements for the host processor in regard to handle the eUniStone.

5.1.2 Host Application

The focus of this part is to provide information on how to control, initialize and configure eUniStone to make the accessory working with major Smart Phones. To design the host application the commands and responses in the SPP-AT specification [2] communication over UART interface are key. Also HW aspects needs to be taken care of e.g. Low Power Mode and HW reset, the reference design and HW is described in this document but also in the HW specification [1].

The below picture shows how a general state machine for the Host application may be implemented and running in the host processor controlling the eUniStone module.

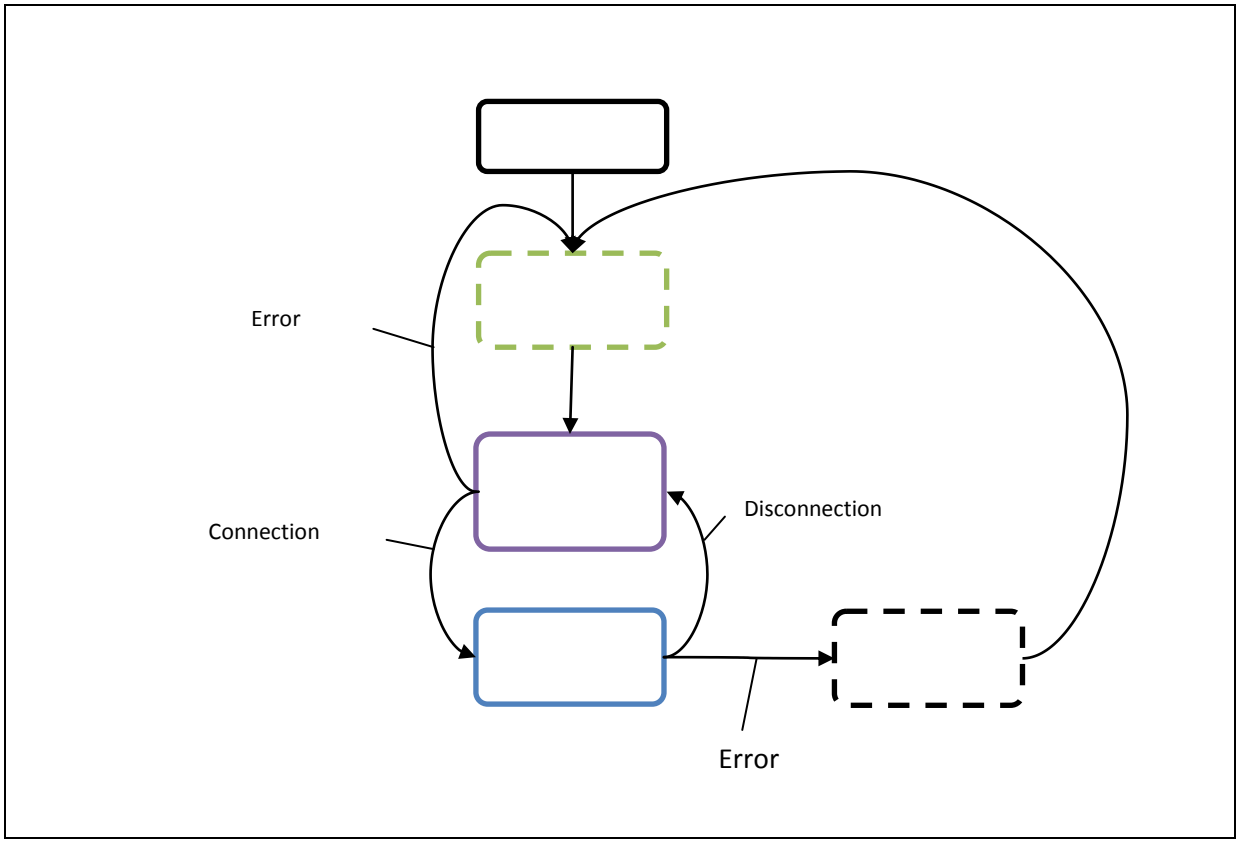


Figure 8. General state machine for a Host application as accessory

Below is a short description of the states from the Host application state machine from above.

1. **Started,** is after power on or a HW reset. To do this the host processor must be able to Control GPIO's for HW reset and Low Power Mode (LPM).

2. **Initialize,** requires sending SPP-AT commands and parsing results. Furthermore, basic configuration, setting security, enabling Page and Inquiry Scan, setting local name etc.

3. **Not Connected,** can be an idle state were it waits for an incoming connection or it waits for the user of the Host application to make an input so that it connects to another device. This state could also be very short if the host application is configured to directly try to and successfully connects to another device.

4. **Connected**, requires that the host application is able to handle transmit and receive of data to and from the remote device (e.g. a mobile phone). Disconnect, is an intermediate state where a disconnection has occurred on purpose or by unknown reasons.

When being in the Not Connected or Connected state it is important to save power. The Not Connected state is most probably the normal state and LPM (Low Power Mode) should be used at all times as long as the host is not sending any commands to eUniStone nor receiving a wake-up from

eUniStone. In the Connected state LPM should be used jointly with having the link in Sniff mode. LPM can also be used when the link is not in Sniff, but there is no power saving benefit since eUniStone has to use the high precision clock in this state.

Since Bluetooth is wireless, a link disconnection can occur due to radio disconnections, and such errors needs to be handled. In case of any error's in any state there should be an error handling routine that takes care of it and initializes eUniStone again. If the error occur while connected to another device it's recommend to try to **Disconnect** the link before making a new initialization.

5.1.3 Smart Phone compatibility requirements

To design the complete accessory and the Host application and to make it compatible to Smart Phones the commands and responses in the SPP-AT specification [2] needs to be well understood and implemented in the Host application. There are various compatibility requirements depending on Smart Phone which is covered in the remainder of this chapter.

- Be a connectable (device B) or connecting (device A)
- Make the right initialization/configuration
- Use correct security settings
- Use correct UUID
- Use CoD as needed
- Register correct services
- Need for certain licensees for iPhone compatibility

The above are the main considerations and specific configuration which has impact on the compatibility vs the three main Smart Phones. Also the actual phone App settings/functionality has impact on the compatibility.

5.1.3.1 Serial Port Profile - device A and B

The device (Accessory or Smart Phone) that starts the Bluetooth Serial Port connection is a "device A" and the device that accept a connection is a device B, this is nomenclature from the SPP specification. eUniStone can be configured both as a device A and B. Normally a Serial Port Profile (SPP) accessory is configured as a device B, providing a service to e.g. a mobile phone.

eUniStone as a connectable device B should be configured, by the Host application using the following SPP-AT commands and their specific functionality [2].

- AT+JSEC (Set security level, variable/fix PIN, Input and Output Capabilities)

- AT+JSLN (Set a local name so that the remote device can find when doing a device discovery or request when connected)
- AT+JRLS (Register up to three services, UUID and CoD)
- AT+JDIS (Enable Page and/or Inquiry Scan)
- AT+JAAC (Set auto accept or host accept of incoming connection)

eUniStone as a connecting device A should be configured, by the Host application with the following SPP-AT commands.

- AT+JSEC (Set security level, variable/fix PIN, Input and Output Capabilities)
- AT+JSLN (Set a local name so that the remote device can request it when connected)
- AT+JCCR (Create connection to a known remote device service)

When creating connections it is necessary to find the remote devices and services first - use then device discovery (AT+JDDS) and service search (AT+JSDS).

The AT+JSEC command may only be sent after power up, a HW or SW reset. It shall therefore always be the first sent command. Local name and services shall always be written for a device B since it will be easier for the connecting device to find it. It's also recommended to do such settings on a device A since after a device A has connected to a remote device the remote device might do a remote name request and add the name to its device list.

5.1.3.2 Security settings – Input & Output Capabilities

Security settings are important and can seem complex to use. It is however necessary to configure, and it has impact on how the connection set-up and bonding may work with Smart Phones. When the security command (AT+JSEC) is used the variable/fix PIN, Input and Output Capabilities are set.

AT+JSEC=<security_mode>,<PIN_type>,<length_PIN_code>,<PIN_code>,<Input_capability>,<Output_capability>

Example: AT+JSEC=4,1,04,1111,2,1

Security mode 4 is the supported mode by eUniStone since it supports Secure Simple Pairing. Variable PIN is normally used when the device have Input and Output Capabilities. Devices that are missing Input and/or Output capabilities can use a fixed PIN that is set with the security command. The Input and Output capabilities are also set. The Input and Output capabilities shall be set according to the capabilities available in the accessory.

Table 3. Configurable combination of Input and Output Capabilities for eUniStone.

Input	Output	Description	Example
0	0	No input and No output	Sensor
0	1	Display only	Display
1	0	Input, no output	Remote control
1	1	Input and Display YesNo	Display with settings
2	0	Keyboard Only	Keypad
2	1	Keyboard, Display YesNo	Mobile phone/Handheld device

For example if the host solution have an input button, the Input capability shall be set to '1'. If the host solution has a keyboard, the input capability shall be set to '2'. The Output refers to the accessories capability like e.g. screen.

When bonding with a Smartphone the accessories configured Input and Output Capabilities used, will result in different bonding procedures since the manufactures of the mobile phones have made interpretation and implementation how it should work. See chapter 5.2

Secure Simple Pairing (SSP) is supported but also the legacy paring is supported when connecting to older Bluetooth devices for backwards compatibility, see SPP-AT specification [2]

5.1.3.3 Registering service - UUID and CoD.

To be able to use eUniStone in an accessory in combination with a Smart Phone App, specific Universally Unique Identifier (UUID) and Class of Device (CoD) are needed. UUID, CoD and also service name and port number is registered in eUniStone with the command AT+JRLS. The **UUID's** that can be used shall either be one containing the Serial Port Profile or a random UUID which is specific for Android phones.

A Serial Port Profile accessory, according to Bluetooth standard, uses either a short or a long UUID as below;

- the short UUID 0x1101 or;
- the long UUID 0x00001101000010008000000805F9B34FB

Class of Device should be set according to what kind of device the accessory is. CoD does not tell the exact services and functionality available in an accessory, but it is recommended to have a class of device in eUniStone as close as possible to what's implemented. The CoD consist of a 24 bit field. When Smart Phones search for accessories the CoD may result in finding or not finding the accessory since the manufactures of the phones handle CoD differently. Below are a couple of examples;

Table 4. Class of Device bits for a handheld terminal with bar-code scanner.

	Major Service Classes											Major Device Classes					Minor Device Class							
Bi	2	2	2	2	1	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0

t:	3	2	1	0	9	8	7	6	5	4	3	2	1	0									
	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0

CoD = 0x000560, details about the bits are available at https://www.bluetooth.org under the section called “Assigned numbers, Baseband”.

Table 5. CoD examples for some accessories.

Accessory	Class of Device	Comment
Sensor	0x001F00	Uncategorized device
Display	0x000610	Imaging, Display
Remote control	0x00050C	Peripheral, Not Pointing device, Remote control
Display with settings	0x000610	Imaging, Display
Keyboard	0x000540	Peripheral, Keyboard
Handheld terminal	0x000560	Peripheral, Keyboard, Handheld scanner
iPhone SPP accessory	0x240704	Audio, Rendering, Wearable, Wristwatch

The CoD is set to 0x240704 in the below example and works with all three Smart Phones in this document, and it is iPhone requiring it for Accessories. In the following examples, the service name is “Serial port” and is set to port 0x01 (possible values are 0x01-0x30), furthermore it starts with length field for UUID and service name.

1. Registering a service with short UUID: AT+JRLS=04,11,1101,Serial port,01,240704
2. Registering a service with long UUID: AT+JRLS=32,11,00001101000010008000000805F9B34FB,Serial port,01,240704

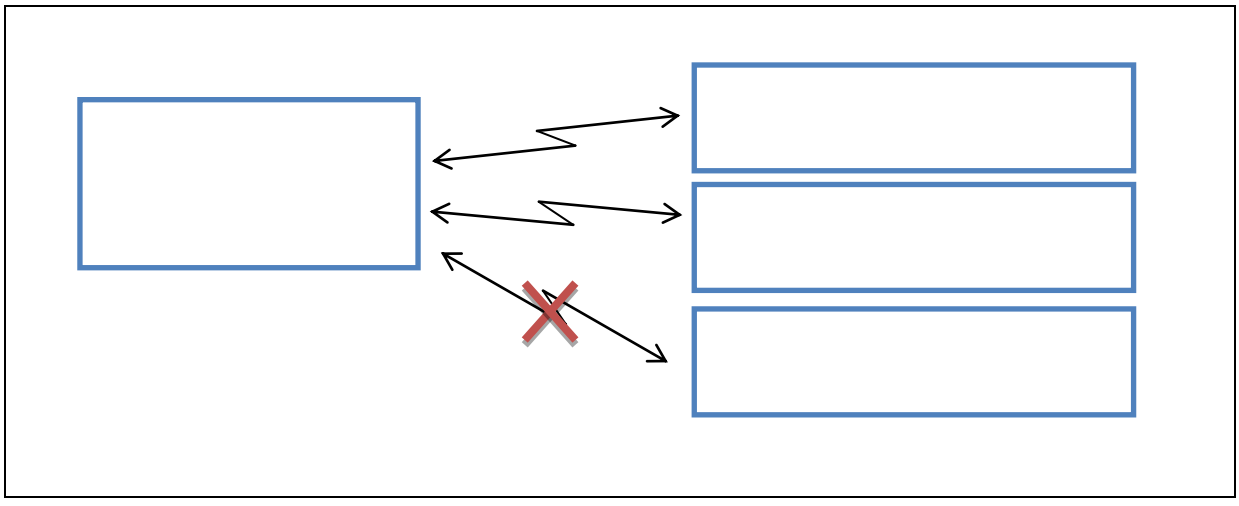


Figure 9. Smart Phone Apps using Serial Port Profile UUID connecting to Serial port accessory.

When the UUID with the Serial Port Profile is used any other device or mobile phone application which use the Serial Port Profile can connect to the Accessory with eUniStone (iPhone require a specific long UUID).

5.1.3.4 Registering multiple services and connection handling

eUniStone supports three registered services. To be able to find eUniStone with Android, Windows phone and iPhone up to three SPP services need to be registered. The two first services in the below example should cover all three phones. The first service, UUID with the Serial Port Profile, is registered on port 0x01 and is used for Android phones and Windows phone. The second can be used by iPhone. Also a third service with a random number UUID has been added. Random means here that the Android Smart Phone App has that specific UUID and thereby will find and connect to a specific accessory carrying that exact UUID. Which services to register in the accessory depends on what App and Smart Phones shall be supported.

1. AT+JRLS=32,13,0000110100001000800000805F9B34FB,Serial port 1,01,240704
2. AT+JRLS=32,13,00000000DECAFADEDECADEAFDECACAFF,Serial port 2,02,240704
3. AT+JRLS=32,13,D2CA6960F6A042308BE980CF63B25BAF,Serial port 3,03,240704

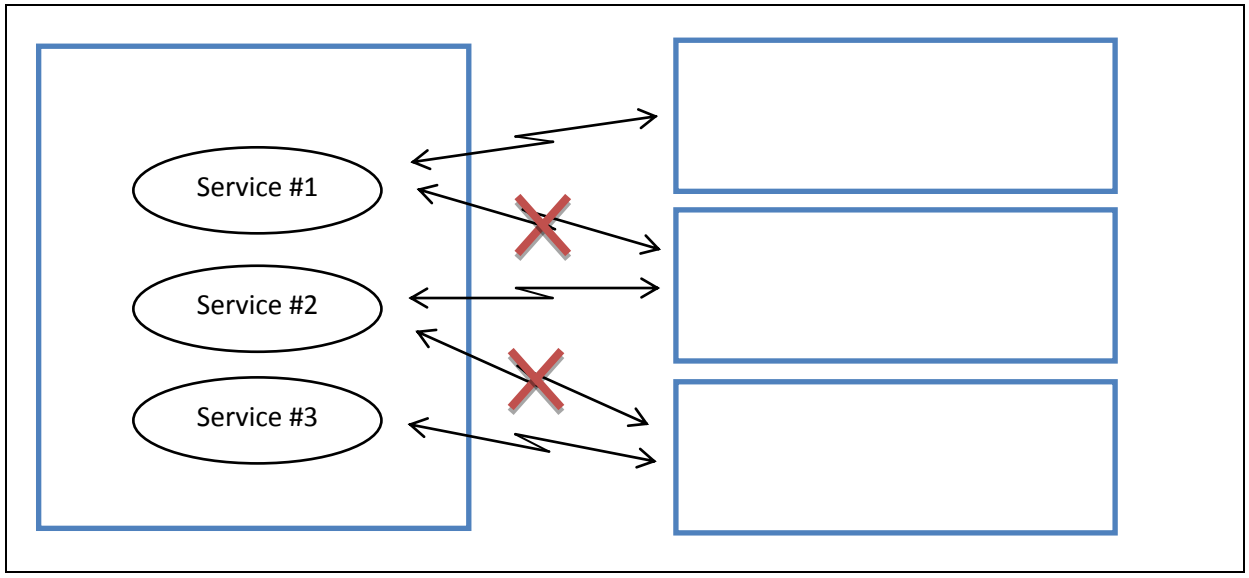


Figure 10. eUniStone with three registered services and possible connections to Smart Phones.

The figure above shows how the various Smart Phones are able to connect, or not connect, to an accessory using eUniStone with three registered services. Observe that only one CoD is possible to store and use, so the last registered will always be used.

When a remote device connects to eUniStone it will tell the host application which service that has been connected to in the Connect Confirm Response. With the services used in the previous chapter the following three successful Connect Confirm Response will be received.

1. +RCCRCNF=500,0000110100001000800000805F9B34FB,0
2. +RCCRCNF=500,00000000DECAFADEDECADEAFDECACAFF,0
3. +RCCRCNF=500,D2CA6960F6A042308BE980CF63B25BAF,0

Depending on which service that is connected the host application can choose to handle the connection with different actions, see Figure 11.

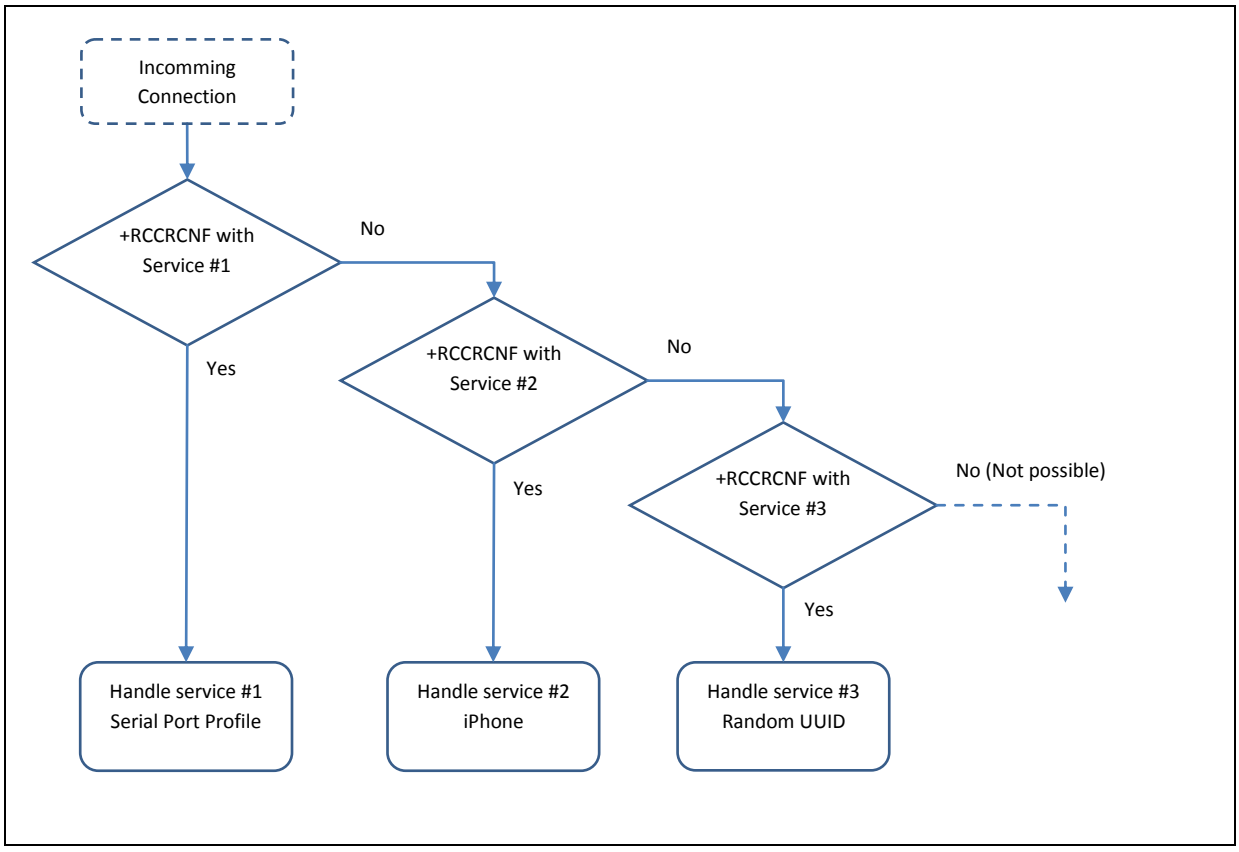


Figure 11. Handling incoming connection depending on service.

For example if the connection contains the first service it can choose to just be a cable replacer. If it contains the iPhone service, it shall also initiate a specific Apple authentication before being able to send and receive data. If it contains the third random UUID (Android specific) it can take a proprietary actions that only is used together with remote devices that has the same UUID.

5.1.3.5 Smart Phone connecting to Accessory

The accessory is normally a device B and the general SPP-AT sequence for enabling the eUniStone to be connected by a Smart Phone is;

Table 6. Accessory (eUniStone) set to connectable, device B

Step	Direction	Command/Response	Note
1	Host →	AT+JRES	Reset of eUniStone
	← eUniStone	ROK	
2	Host →	AT+JSEC=4,1,04,1111,1,1	Enable Security with Input capabilities: Button Output capabilities: YesNo

Step	Direction	Command/Response	Note
	← eUniStone	OK	
3	Host →	AT+JSLN=09,eUniStone	Write Local Name
	← eUniStone	OK	
4	Host →	AT+JRLS=32,11,000011010000100080000080 5F9B34FB,Serial Port,01,000000	Register Local Service, Serial Port with number 0x01, name “Serial Port” and CoD 0x000000
	← eUniStone	OK	
5	Host →	AT+JAAC=1	Auto accept connections
	← eUniStone	OK	
6	Host →	AT+JDIS=3	Enable Inquiry and Page Scan, this is how a device B gets connectable.
	← eUniStone	OK	

For a Smart Phone to start the connection and get connected to the accessory with above settings the normal procedure is;

On the phone (will vary between Smart Phones):

1. Go to the Bluetooth settings or start the application.
2. Enable Bluetooth and search for nearby devices.
3. Choose the correct device.

On Accessory and phone:

4. A passkey will be displayed on both the accessory and on the phone. Compare them and, if they are equal, accept the connection on both accessory and phone.

The procedure will vary and depend on the UUID, CoD and Input, Output Capabilities set with the security command on eUniStone. See chapter 5.2 for detail about Android, Windows Phone and iPhone in combination with settings on eUniStone.

5.2 Smart Phone specifics

The various Smart Phones like Android, Windows mobiles and iPhone use their own OS’s and software and hardware implementations from various vendors. The major Smart Phones and operating systems today are;

- Android by Google used by e.g. Samsung
- Windows Phone 8 by Microsoft, used mainly by Nokia
- iOS by Apple

Since the Smart Phones uses different operating systems they need their own App implementations to connect to an accessory. Also the behavior and requirement of the Smart Phones may differ depending on both operating system and version. Included here are the basics for securing easy set-up and compatibility with the major Smart Phones and OS's.

5.2.1 Android Smart Phones

To keep up compatibility with Android Smart Phones using specific Bluetooth Apps there are a couple things to keep in mind both for the Accessory and App development. The below is valid for 4.x versions and possibly also older versions of Android but it cannot be guaranteed.

5.2.1.1 Registering service for Android compatibility

An Android application that shall work with an Accessory (eUniStone) solution can either use a UUID containing the **Serial Port Profile UUID** or a it can use the **random UUID** which only is used for accessories (or other Apps) connecting to specific Android Apps. In either case it is important that the chosen UUID is registered in eUniStone and used in the Android application. It is recommended to use a CoD other than 0x000000, since some Smart Phones with specific Android versions, will require this.

The **Serial Port Profile UUID**, CoD, service name and port number of the service is registered with the command AT+JRLS, and the values shall be written to eUniStone as initialization. The below example should work with most Android Smart Phones. The service is called "Serial port" and set to port 0x01. The name and port number can be chosen to be different to the example.

- Registering a UUID containing the serial port profile:
  AT+JRLS=32,11,000011010000100080000080 5F9B34FB,Serial port,01,240704

What CoD to choose depends on the specified behavior of the accessory. However, in some cases it is also important to use info from the field on what works. Details about CoD can be found on https://www.bluetooth.org under the section called "Assigned numbers, Baseband".

Random UUID can be used if the Accessory (eUniStone) only shall work with a specific Android Smart Phone App. If eUniStone uses a random UUID, an Android Smart Phone App needs to be developed which use the same specific random UUID. The specific accessory with eUniStone and the Android Smart Phone App will then more easily be associated in a way so that this Android Smart Phone App will only connect to the accessory using that specific random UUID.

- Registering a random UUID:
  AT+JRLS=32,11,D2CA6960F6A042308BE980CF63B25BAF,Serial port,01,240704

Observe that the long random UUID is only generated randomly once during development of the accessory and Smart Phone App. Most programing languages and tools contain functions for generating a long random UUID. UUID generators can for example be found in Java. Also there are online webpages that can generate random UUID's.

A general mobile solution with the Serial Port Profile will not be able to connect to this kind of specific accessory. See below the example of working and non-working connections.

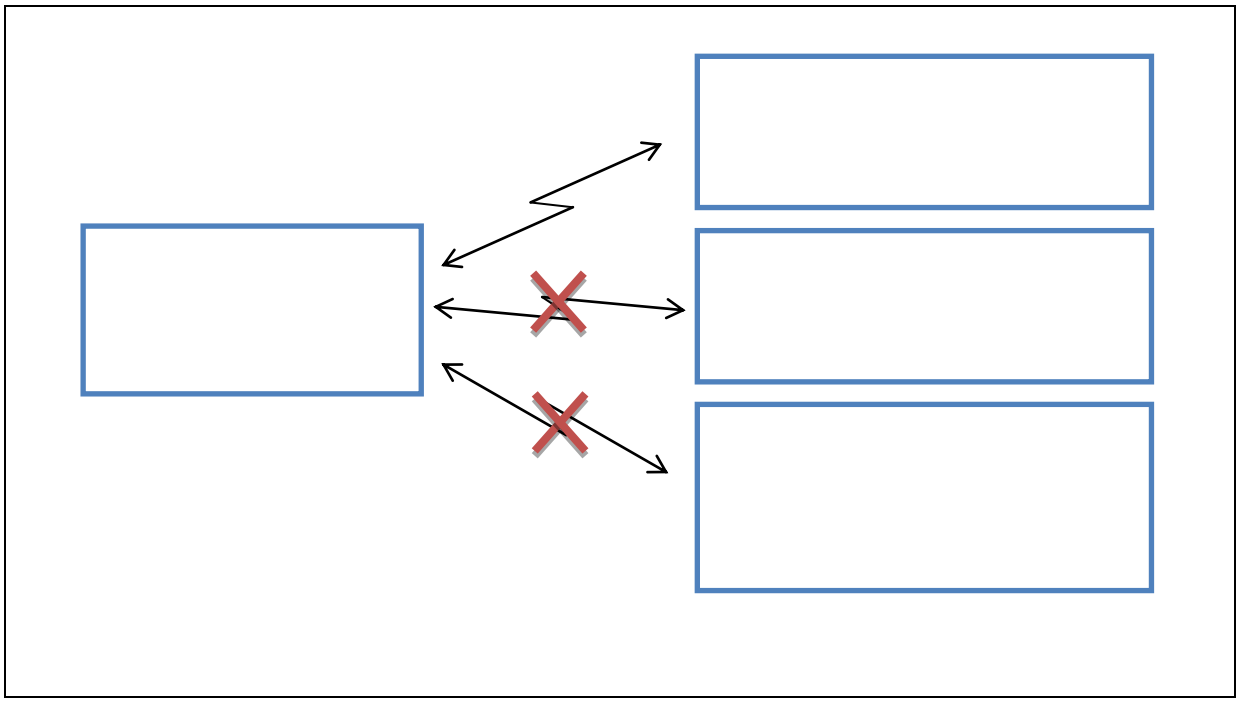


Figure 12. Accessory and Android Smart Phone App using the same UUID will be able to connect.

5.2.1.2 Security settings and connecting/bonding to Smart Phone versus App in the phone

Bonding an Android phone with Accessory (eUniStone) can be performed either through the Bluetooth settings in the Android phone or directly through an App running on the phone. To be able to do it inside, the App needs to support such a procedure. The security command (AT+JSEC) is used to set the security for eUniStone, variable/fixed PIN and the Input and Output Capabilities. The Input and Output Capabilities are set according to the capabilities of the accessory. Depending on which capabilities are set the bonding with Android Smart Phones or a specific App has minor differences. The below tables show expected results.

Table 7. Android Smart Phone - bonding with eUniStone

Android		eUniStone		
Input	**Output**	**Input**	**Output**	**Comments**
2	1	0	0	Bonding is accepted on phone only. [1)]
2	1	0	1	Pass key is shown by both eUniStone and phone. Bonding is accepted on phone only.
2	1	1	0	Bonding is accepted on phone only. [1)]
2	1	1	1	Pass key is shown by both eUniStone and phone. Bonding is accepted on both eUniStone and phone.

2	1	2	0	Pass key is shown on phone. Bonding is accepted by pass key input on eUniStone
2	1	2	1	Pass key is shown by both eUniStone and phone. Bonding is accepted on both eUniStone and phone.

[1)] Inside an App the bonding and connection setup might not be possible with input, output set to 0, 0 or 1, 0. The reason for that is that the App is requesting a “secure RFCOMM socket” with man-in-the-middle protection. To be able to use the input, output combinations 0, 0 and 1, 0 on eUniStone the App should request an “insecure RFCOMM socket”.

When doing Secure Simple Pairing a bonding procedure is performed and link keys are made between the two devices. Next time the devices connect, the connecting procedure will be easier and quicker, since there are already valid link keys to be use by the two devices (Smart Phone and eUniStone).

With the above information and other supporting references it should be possible to in a fast and secure manner develop a Bluetooth accessory, using eUniStone, and thus being compatible with Android Smart Phone application.

5.2.1.3 Android market and Developing Apps for Android Smart Phones

To publish or download Android Apps the following sites are normally used. https://play.google.com/store and https://play.google.com/apps/publish. However, it is possible to download Android Apps directly from other sources as well.

Developing applications for Android phones can be done using e.g. Eclipse [11]. Information on how to download and install the Android SDK is found on http://developer.android.com. The main programing language used is Java, C and C++ can also be used. This chapter describes the most common classes and methods for settings up and managing a Bluetooth connection in Android. More about Android App development is available in the Demo Specification [15].

5.2.2 Windows phone

Developing Accessories supporting Bluetooth communication for Windows Phone Apps resembles very much with the Android compatibility procedures. However, the random UUID concept is not used and information and examples are available but rare in this community yet.

5.2.2.1 Registering service for Windows Phone

Windows phone will require that the UUID contains the Serial Port Profile to connect with Accessories (eUniStone). Either the short 0x1101 or the long 0x00001101000010008000000805F9B34FB.

- Registering the short: AT+JRLS=04,11,1101,Serial port,01,240704
- Registering the long: AT+JRLS=32,11,00001101000010008000000805F9B34FB,Serial port,01,240704

The Class of Device, CoD, written to eUniStone should also for a Windows Phone accessory reflect the functionality and features of the accessory as described in previous chapters, to be compatible with all three Smart Phones the CoD used should be 0x240704.

5.2.2.2 Bonding

Bonding a Windows Phone with eUniStone can be performed either through the Bluetooth settings in the Windows Phone or directly through an application running on the phone. To be able to do it inside an application, the application needs to support such a procedure. Depending on which capabilities set with the security command (AT+JSEC), the bonding with Windows Phone and eUniStone will vary according to the comments in below table.

Table 8. Windows Phone - bonding with eUniStone.

Windows 8		eUniStone		
Input	**Output**	**Input**	**Output**	**Comments**
2	1	0	0	Just Works with no user input on either side.
2	1	0	1	Just Works with no user input on either side.
2	1	1	0	Just Works with no user input on either side.
2	1	1	1	Pass key is shown on both eUniStone and phone. Bonding is accepted on both eUniStone and phone.
2	1	2	0	Pass key is shown on phone. Bonding is accepted by pass key input on eUniStone
2	1	2	1	Pass key is shown on both eUniStone and phone. Bonding is accepted on both eUniStone and phone.

5.2.2.3 Windows Phone Store, development and design

This is the site for providing Windows Phone Apps to the public: http://www.windowsphone.com/en-us/store.

Install windows phone SDK, it can be downloaded from the Microsoft website: (http://dev.windowsphone.com/en-us/develop)

- Visual Studio 2010 Express for Windows Phone (IDE)
- Windows Phone emulator (To test windows phones app, to use hardware related apps we will need a real phone)
- XNA Game Studio 4.0 (For Game development)
- Silverlight (Basic GUI development, uses XAML-based application development)
- Expression Blend for Windows Phone (It has more feature for GUI building, not necessary to use)

- .NET Framework 4

To start developing basic windows phone application open visual studio and click **File --** click **New Project,** it with show you a dialog box, select **Windows Phone Application**

5.2.2.4 Programming language for Windows Phone

The programming primarily use for Windows phone development is C#, but other programming languages from Microsoft can be as well like VB.net, Visual C++.

For Bluetooth communication Windows Phone provides a basic API, It has four basic classes which are

- PeerFinder
- Peerinformation
- StreamSocket
- ConnectionRequestedEventAArgs

If you are trying to make a connection between App to App you will need to add the capability "ID_CAP_PROXIMITY" and for App to device, add the capabilities ID_CAP_PROXIMITY and ID_CAP_NETWORKING. If you don't add these, you will get an exception.

Connecting to a device (Code snippet)

```
PeerFinder.AlternateIdentities["Bluetooth:Paired"] = "";

var peerDevices = await PeerFinder.FindAllPeersAsync();
```

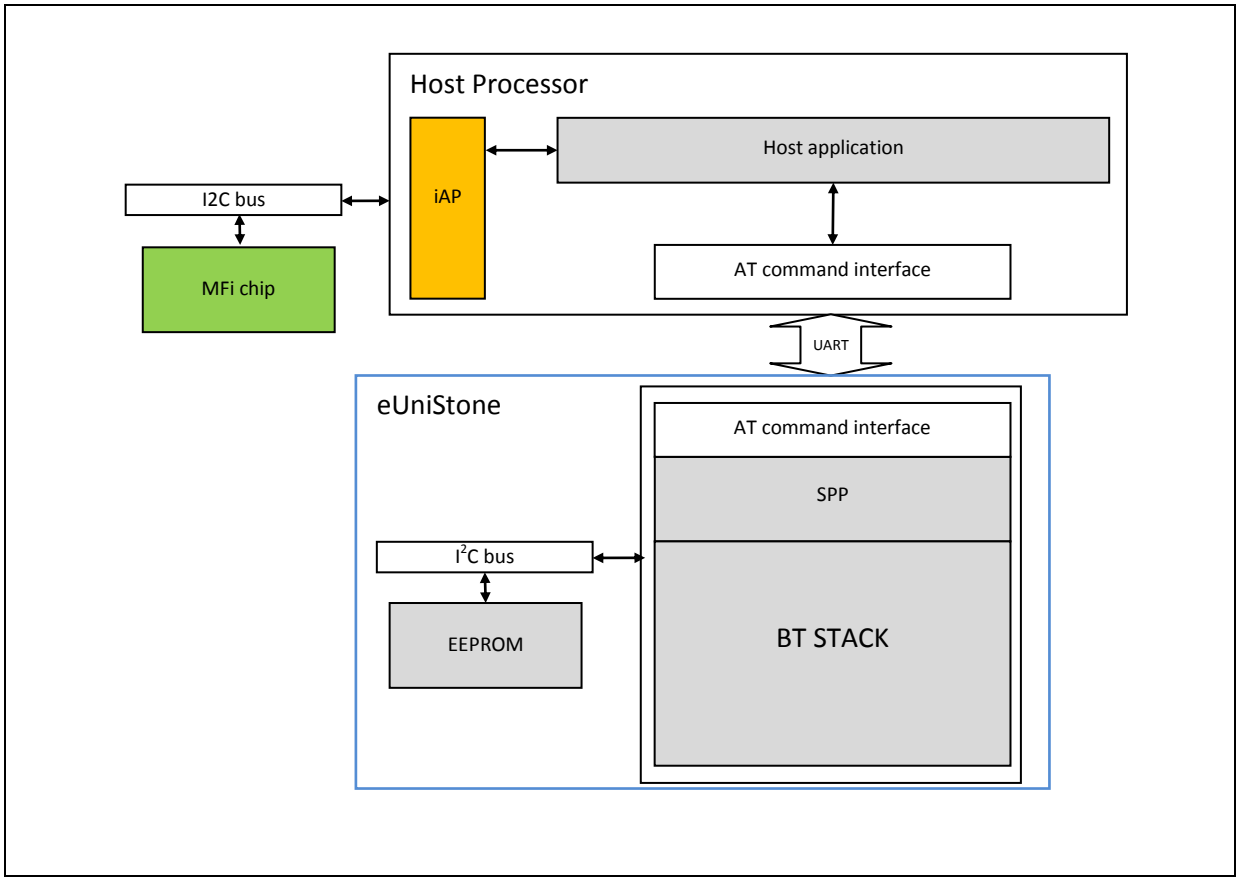
When you get paired device you get information like Hostname, address etc.

```
PeerInformation device = peerDevices[0];

var hostname = device.Hostname;
```

5.2.3 iPhone

This chapter describes how the eUniStone PBA31309 Bluetooth module may support communication with the Apple iPhone/iPad/iPod touch products. The iPhone/iPad/iPod touch products, hereafter referred to as Apple devices, implement an additional protocol layer on top of RFCOMM called iAP (iPod Accessory Protocol). The eUniStone host, hereafter referred to as the accessory, must identify and authenticate itself towards the Apple device to gain access to iAP



The above hardware and software is needed in the accessory to be compatible and working with iPhone applications. Furthermore, developing Accessories using the Bluetooth Serial Port Profile and applications for iPhone and iPads, important considerations must be taken on how to make the accessory compatible with iPhone Apps.

- UUID, must be a specific long version
- CoD, only 0x240704 works due to historical reasons
- MFi developers or manufacturing licensing from Apple

5.2.3.1 Registering service for iPhone

The iPhone will require eUniStone to use the UUID 0x00000000DECAFADEDECADEAFDECACAFF and CoD 0x240704 or similar see below. In the below string the above UUID and CoD is used when registering the service in eUniStone.

- AT+JRLS=32,11,00000000DECAFADEDECADEAFDECACAFF,Serial port,01,240704

The eUniStone Class of Device (CoD), set by the AT command AT+JRLS, shall be set to a specific value in order to be discovered by an Apple device. Using e.g. CoD 0x240704 for the Accessory it will in many cases not reflect the correct functionality of the device since this means that it is a (Audio, Rendering, Wearable, Wristwatch) device.

The Major Service Class shall be set for Audio (bit 21) and Rendering (bit 18). The Major Device Class shall be set for either Audio/Video or Wearable. The Minor Device Class shall be set to one of the following:

- For Audio/Video:
  - Wearable Headset Device
  - Hands-free Device
  - Loudspeaker
  - Headphones
  - Portable Audio
  - Car Audio
  - HiFi Audio Device
- For Wearable:
  - Wristwatch
  - Jacket
  - Helmet
  - Glasses

Details about the assigned numbers for the various applications are found at the Bluetooth.org. Additionally when an iPhone application is connecting to eUniStone the iAP authentication needs to be handled by the accessory. Full insight in iPhone accessory is available to MFi licensees.

5.2.3.2 Bonding

Bonding an iPhone with eUniStone is always performed using the Bluetooth settings in the iPhone. After bonding, it is possible to start an iPhone application and connect to eUniStone. When the iPhone application connects to eUniStone the iAP authentication will be started and handled automatically by iOS. The accessory needs to contain an MFi chip to be able to handle the iAP authentication. Depending on which capabilities set in Accessory (eUniStone) with the security command (AT+JSEC), will be according to the below table.

Table 9. iPhone - bonding with eUniStone.

iPhone		eUniStone		
Input	**Output**	**Input**	**Output**	**Comments**
2	1	0	0	Bonding is accepted on iPhone only.
2	1	0	1	Pass key is shown on both eUniStone and iPhone. Bonding is accepted on iPhone only.
2	1	1	0	Bonding is accepted on iPhone only.
2	1	1	1	Pass key is shown on both eUniStone and iPhone. Bonding is accepted on iPhone only.
2	1	2	0	Pass key is shown on iPhone. Bonding is accepted by pass key input on eUniStone
2	1	2	1	Pass key is shown on both eUniStone and iPhone. Bonding is accepted on both eUniStone and iPhone.

Observe that the bonding between iPhone and eUniStone in Table 9 only shows the Bluetooth bonding and does not involve the complete iAP authentication which is also required when an application running on the iPhone connects to the Accessory (eUniStone).

5.2.3.3 iOS and MFi license

To be able to develop applications for iOS you need to become an Apple developer. Information about the program and registration as Apple Developer can be found on https://developer.apple.com/devcenter/ios/ App development is done using Apple Xcode software. The programing language used in Xcode is Objective-C, C and C++.

To be able to use iAP and the MFi hardware you need to enroll into the MFi program at Apple's development site for MFi. You will find the MFi program on the following link https://mfi.apple.com On that page click the link "Apply now" and follow the instructions. You can either apply for a developer license or a manufacture license. More information on how to apply and which license that fits your need can be found on http://mfi.apple.com/faqs

6 Reference Design Schematic

The reference design schematic is shown in Figure 13.

- VSUPPLY, VDDUART and VDD1 can be supplied by the same 3.3 V voltage.[1]
- C1 is only need to be placed in case noise is present from the power supply.
- All VSS pins must be connected to ground.
- All NC pins are internally not connected and can be left open.
- Since the internal LPO (low power oscillator) is used by the module then the P1.5/CLK32 pin can be left open.
- Is strongly suggested having test point on P0.12/SDA0 and P0.13/SCL0. These can be useful for debugging purpose.
- A test point on P0.1 or P0.8 is needed for crystal calibration in case the pre-stored value is lost and for the HCI application for RF Testing.
- The line UARTRXD and UARTCTS must remain high during low power mode. If the host cannot drive them all the time, a pull-up might be needed.
- For debugging, test points on the UART lines can be helpful. .
- If JTAG interface is not used, JTAG# pin can be kept open (internal pull up). To enable JTAG interface, a 4.7 kΩ pull down resistor must be put on this pin.
- RESET# pin shall be driven by the host. A pull-down of 4.7kOhm is strongly recommended to reset the device when the power supply fails. See section 3.1 Power up Sequence.
- ONOFF pin shall be connected to VSUPPLY, if it is not used.[2]
- If LPM wakeup input P0.14 is not used, it shall be pulled up to VDDUART.

[1] If only one voltage supply is used, then the ONOFF pin may not be used. The ONOFF pin shall be connected to VSUPPLY.

[2] No reference level or input signal shall be applied to the module while ONOFF is low. Output signal levels are not defined while ONOFF is low.

- If LPM wakeup output P0.0 is not used, it can be left open.
- The GPIO pin P0.1 (pin E5) can be used to indicate the connection status. P0.1 is configured as input pin by default. To use this feature the host must send the AT command “AT+JGPC=FFFD,0000,0000,0000,FFFD” which configures P0.1 as an output pin.

Above points are summarized in Table 10.

Table 10. Default Pin Configuration

Interface	Pin	Name	Note
UART	F4	P0.14	If LPM is not used, this pin shall be pulled up to VDDUART.
UART	E4	P0.0	If LPM is not used, this pin may be left open.
UART	E6	UARTRXD	A 4.7 kΩ pull up resistor can be needed to keep level in LPM.
UART	F6	UARTCTS	A 4.7 kΩ pull up resistor can be needed to keep level in LPM.
	A8	P1.5/CLK32	Internal LPO is used; this pin can be left open.
	B9	P0.15/SLEEPX	Not used. Leave open.
	E5	P0.1	Indicating connection status, when configured as an output pin.
	B5	ONOFF	If not used, connect to VSUPPLY. [2]
	A3	RESET#	Shall be controlled by host I/O.
JTAG	C3	JTAG#	If JTAG interface is not used, this pin can be kept high; otherwise a 4.7 kΩ pull down resistor shall be used to enable JTAG.
JTAG	B3	TMS / P1.0	If JTAG enabled: 4.7 kΩ pull up; otherwise: leave open.
JTAG	D3	TCK / P1.1	If JTAG enabled: 4.7 kΩ pull up; otherwise: leave open.
JTAG	F2	TDI / P1.2	If JTAG enabled: 4.7 kΩ pull up; otherwise: leave open.
JTAG	B3	TDO / P1.3	If JTAG enabled: 4.7 kΩ pull up; otherwise: leave open.
JTAG	B4	RTCK / P1.4	If JTAG enabled: 4.7 kΩ pull down; otherwise: leave open.
JTAG	C4	TRST#	If JTAG enabled: 4.7 kΩ pull down; otherwise: leave open.

In case of using JTAG, the resistors shall be placed on the external JTAG connector.

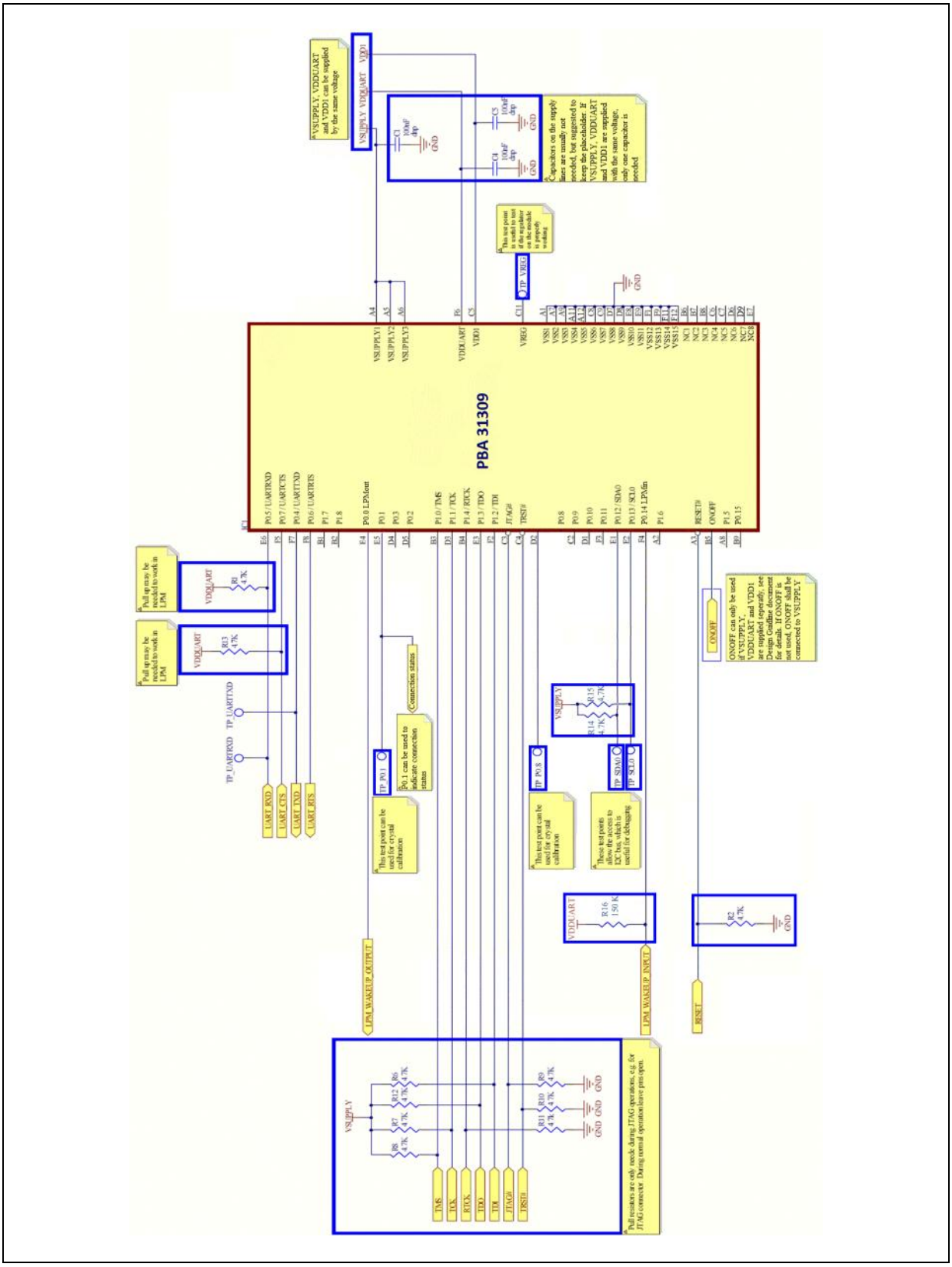


Figure 13. Reference Design

7 Layout

7.1 Two layer PCB reference design

Since eUniStone PBA31309 have an onboard antenna there are only minor considerations for a two layer PCB layout.

- The PCB layers under the antenna shall not contain any metal.
- Place module with the antenna facing the edge of the PCB.
- Do not place the other sides of the module too close to the edge of the PCB.

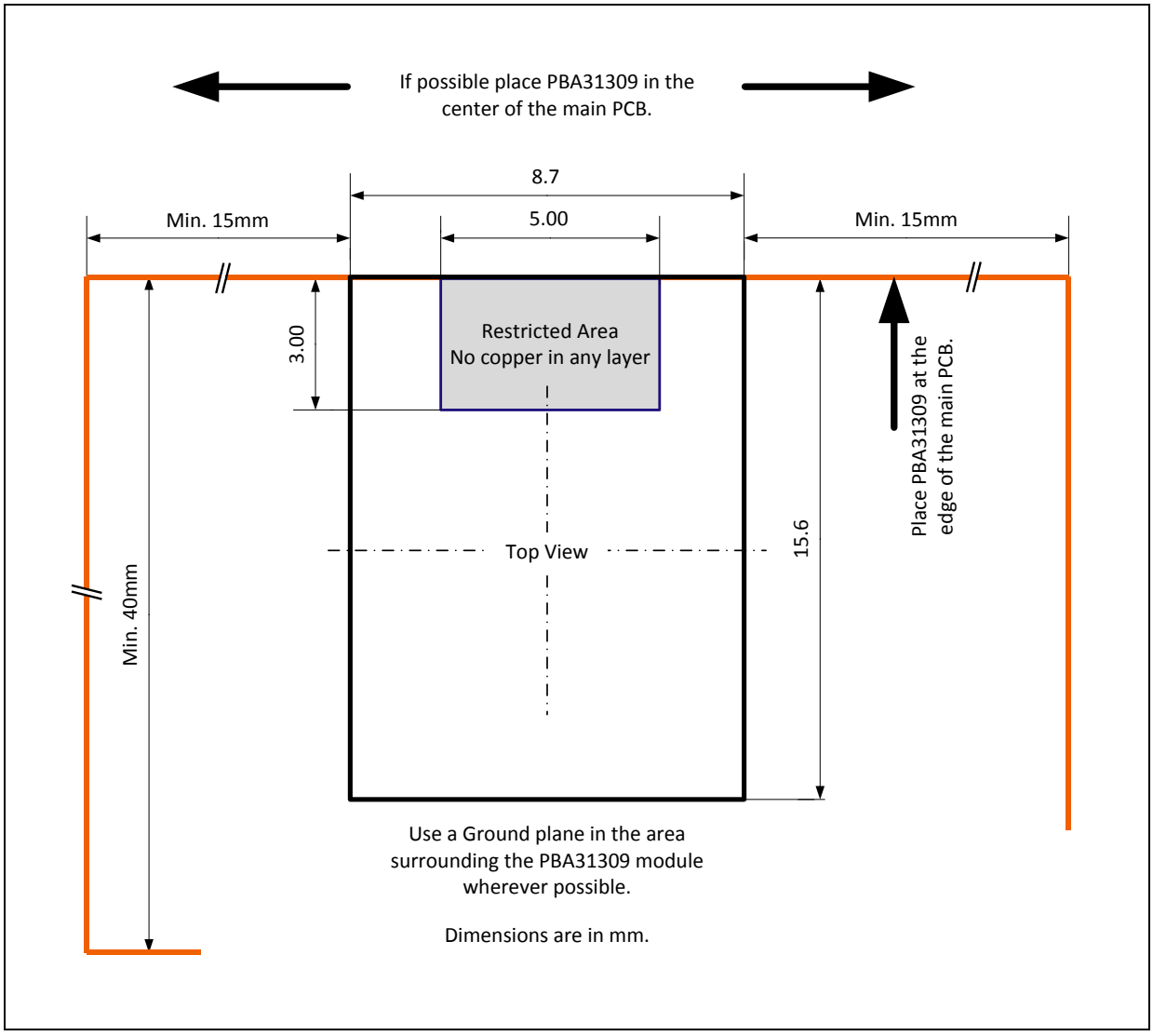


Figure 14. Restricted area under antenna and recommended placement on PCB.

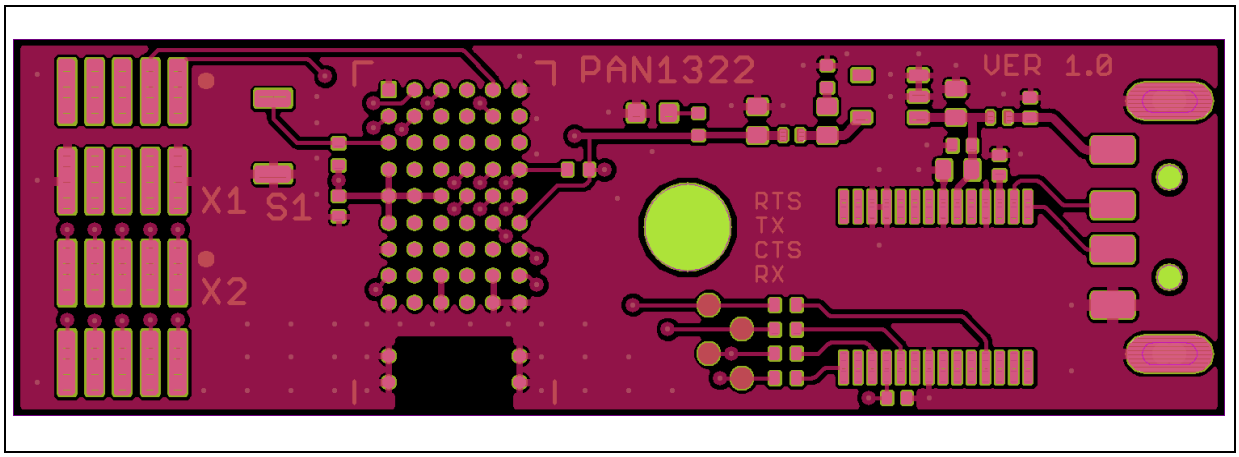


Figure 15. Top layer and drill holes of the eUniStone USB Dongle.

On the top layer eUniStone is placed according to the recommendations in Figure 14. Observe that there is no copper layer under the antenna of the module. A ground plane is spread all over the top layer making a good ground between all components. The power feeding lines have been made wider than the signal lines. The signals are routed to through via, which are the cheapest kind of via and possible to use since there is enough space between the pads of the module.

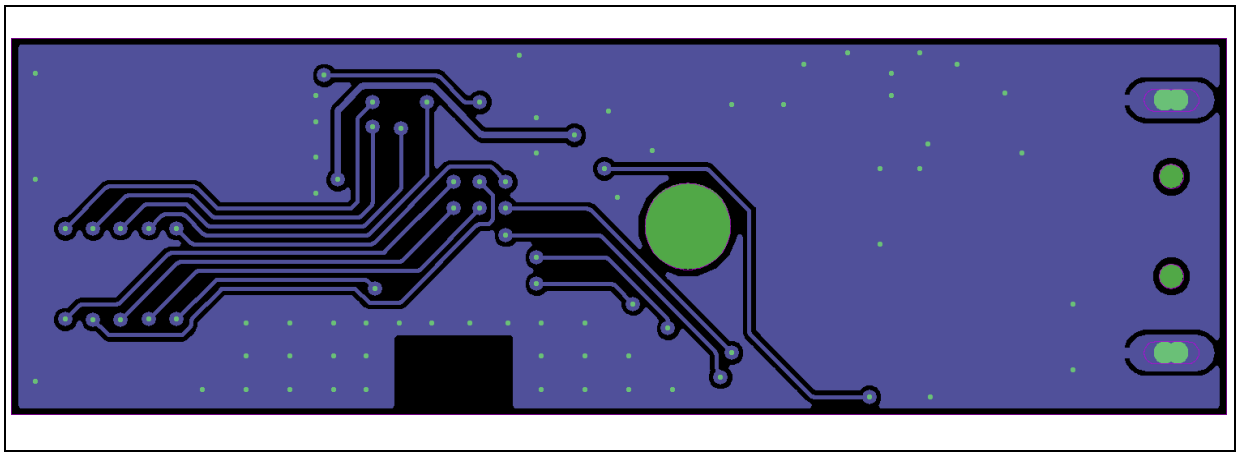


Figure 16. Bottom layer (from above) and drill holes of the eUniStone USB Dongle.

Most of the signal lines are routed on the bottom layers. Also a ground plane is spread, connected with via to the top layer ground.

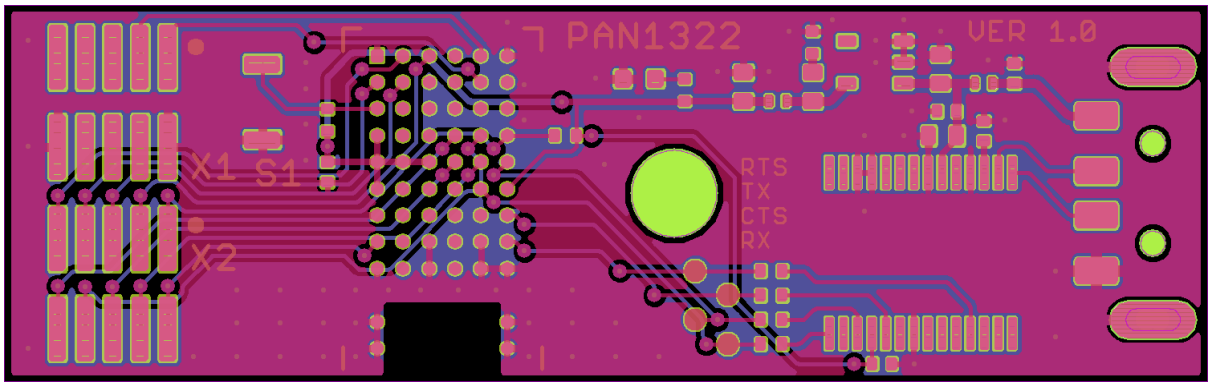


Figure 17. The two layers of the eUniStone USB Dongle.

Observe that there is no copper in any layer under the antenna.

7.2 General four layer PCB design

This chapter describes a general four layer PCB design aimed for more complex cards which have host processors connected to multiple device and possible also clock signals, address and data buses.

Layout design can be divided into the following phases:

- Phase 1: layers assignment.
- Phase 2: components placement.
- Phase 3: routing.

7.2.1 Phase1: Layer Assignment

A correct assignment of the layers may avoid any RF issues. And can make the routing a lot easier.

Figure 18 shows the layers of a general four layer design. In this case, because the number of connections is very small, no special care is required in this phase:

- The top layer is reserved for components' placement.
- Mid layer 1 is used as RF ground and to route some traces
- Mid layer 2 is used for power planes.
- Bottom layer is used for general grounding and to route the rest of the traces

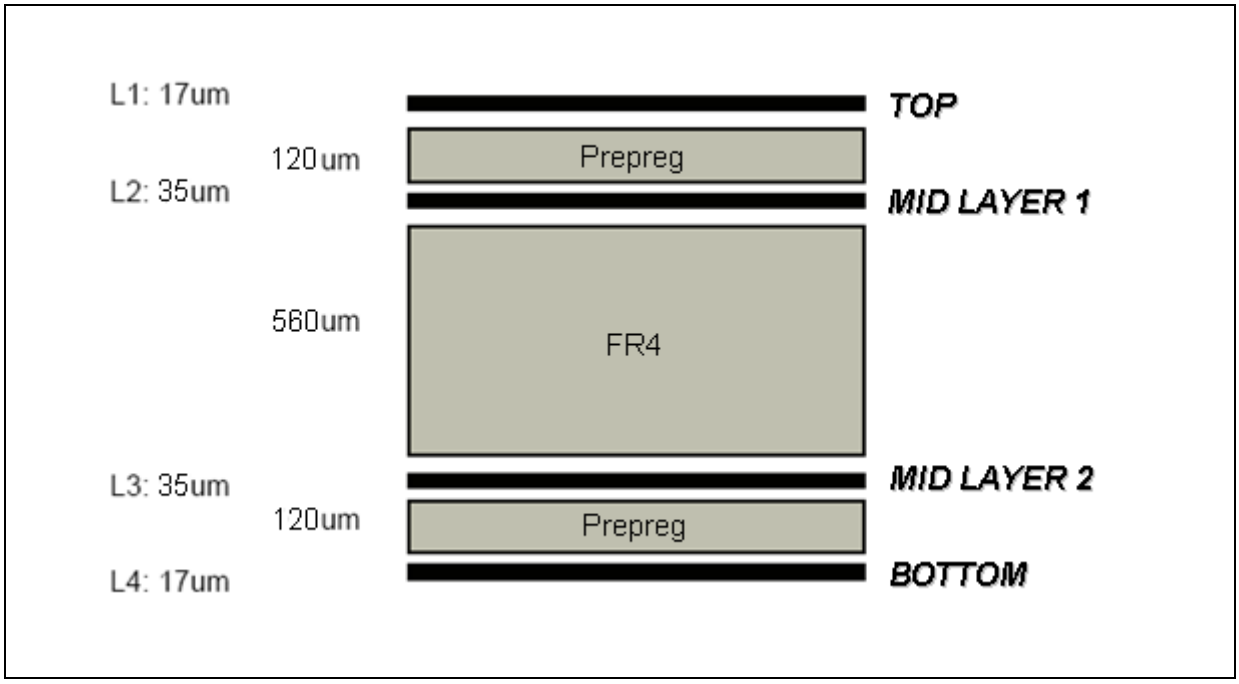


Figure 18. Example of a four layer stack-up

7.2.1.1 Via Holes

Before starting with routing, the layout engineer must know what kinds of via are available. The choice depends from budget considerations: micro via are the most expensive, through via are the cheapest, the others are in the middle.

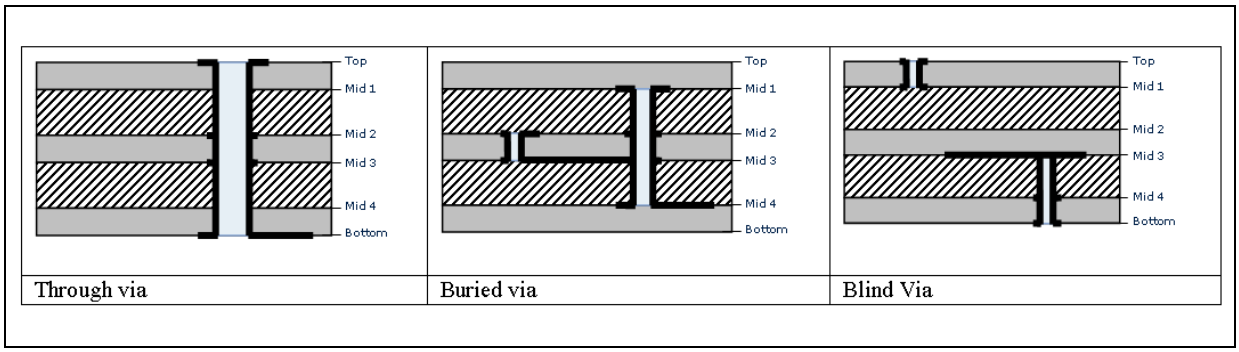


Figure 19. Via Types

Micro via: they have a diameter between 50 µm and 100 µm and can be placed directly into the pads of BGA and similar components. The availability of this technology must be verified with the manufacturing factory. Because of the small diameter much PCB's space can be saved and power and ground planes are less discontinuous.

Buried via: they are connecting two or more inner layers (for example from 2nd to 3rd layer). The diameter is ≥ 100 µm. Since is not required to drill all the PCB they can help to save space for routing.

Blind via: they are drilled from an outer layer to one of the inner layers. Blind via are most efficient if used with buried via. The diameter is ≥ 100 µm.

Through via: they are drilled from the top layer to the bottom layer, and have a typical diameter of 200-300 µm, all the inner layers can be connected to the via. Often are used to connect all the ground areas in the different layers.

When a trace that carry high current moves from one layer to another, more parallel via are required. Placing more via reduce the resistance of the overall connection.

Too high resistance in the power trace might lead to a not constant supply in TDMA system where the TX section is switched on only during the TX slot: when the power section is on, the IC is drawing high current, which produce a voltage drop in the trace, the supply is AM modulated, which ultimately might affect all the performances of the IC.

7.2.2 Phase 2: Components Placement

Placement of the components must be done following the order:

1. Place the connectors in the most rational way considering power distribution and interfaces.
2. Place the eUniStone module with the antenna on the edge of the PCB.

Place all the other components in order to minimize track length and –if possible- keeping separate high frequency and baseband sections.

7.2.3 Phase 3: Routing

7.2.3.1 Basic Hints

Here are some basic hints on routing to avoid crosstalk. These are golden rules not only applicable to eUniStone, but to all designs:

- Group and route traces according to their functionality. Start routing high frequency lines and other sensitive lines.
- Minimize the length of parallel routed traces, in the same layers and in adjacent layer. In adjacent layers orthogonal routing – alternating horizontal and vertical routed layer – is helpful to avoid capacitive coupling between traces on different layers.
- Keep enough separation between traces.

7.2.3.2 Layout Specific Hints

Mid layer 1 should be used as a ground plane:

a. Ground plane must be a continuous plane as large as possible, avoid to just “bring” the ground where is needed.

b. eUniStone has several pins that must be connected to ground. Ideal would be to connect each pin with the ground underneath through via directly on pin. In case this cannot be done a trade-off must be found.

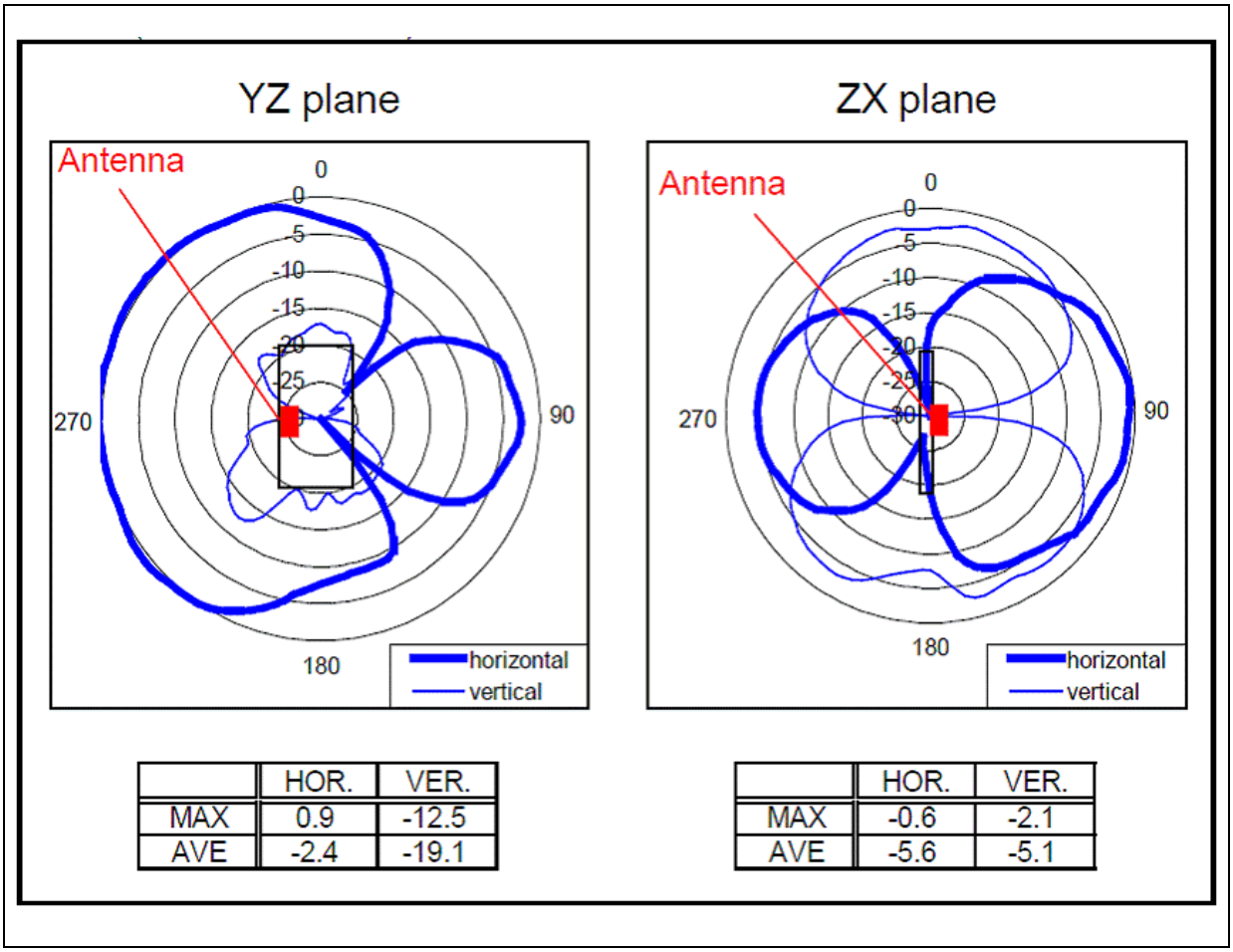
c. Remove dead copper. Small ground filling area, not connect to ground through via, can have unwanted effects. They can behave as antennas, or like a coupling filter or other kind of unwished effects. For this reasons, if it is not possible to connect to ground with enough via they shall be removed.

8 Antenna

The manufacturer of the antenna is Murata. The model number is LDA21K.

The antenna having 50 ohm impedance has been matched on the module to have good efficiency, and enabling the full power of the Bluetooth chip. The matching also assures that the best receiver sensitivity is achieved.

Even if the antenna is onboard of the module there are some considerations to make for the PCB layout, see chapter 7.



	HOR.	VER.
MAX	0.9	-12.5
AVE	-2.4	-19.1

	HOR.	VER.
MAX	-0.6	-2.1
AVE	-5.6	-5.1

Figure 20. Radiation patterns of the built in antenna.

9 Test and Development Tools

This chapter is aimed to clarify some common issues from the customers.

9.1 Tools available for development and testing

9.1.1 SPP Toolbox

The SPP Toolbox is used to control eUniStone with AT commands, to setup links and as described in later chapter, to do RF tests. A detailed manual is delivered with the Toolbox.

For testing of the PBA31309 with SPP AT SW 3.x the SPP Toolbox shall have version number 3.02 or later.

9.1.2 SPP Test Tool

The SPP Test Tool is used for throughput tests, both in command mode and stream. For more details about this tool, please see the manual delivered with the tool.

For testing of the PBA31309 with SPP AT SW 3.x the SPP Test Tool shall have version number 2.1.1.0 or later.

9.1.3 HCI Lite

HCI Lite is used together with eUniStone when it has been loaded with Host Controller Interface software (HCI_TL_ROM_115kbaud.eep). The purpose of the tool is to use it for advanced RF tests. A range of HCI commands is implemented in the tool to be able to set-up eUniStone in different RF modes.

For testing of the PBA31309 with SPP AT SW 3.x the HCI Lite should have version number 3.03 or later.

Attention: After loading Host Controller Interface software (HCI_TL_ROM_115kbaud.eep), the device can only revert to the original SW by the use of an I2C programmer!

9.1.4 eeprog – Aardvark

The eeprog.exe software is used together with the Aardvark I2C/SPI Host Adapter to download software to eUniStone through the I2C interface.

The Aardvark I2C/SPI Host Adapter is a hardware dongle available through Total Phase. The eeprog.exe is only available in one version and is not dependent on SPP AT SW version.

9.2 Setup of Device Under Test (DUT)

Doing RF test with a test instrument like the R&S CBT or similar, controlling testing over the air, can be performed with the SPP software using the SPP Toolbox.

After attaching the device to the computer and opening COM port, follow the below sequence.

1. Enable production mode
2. Enable Device Under Test
3. Read BD-data (Note the BD-address to be used for the connection of the CBT)
4. Search and connect with the CBT

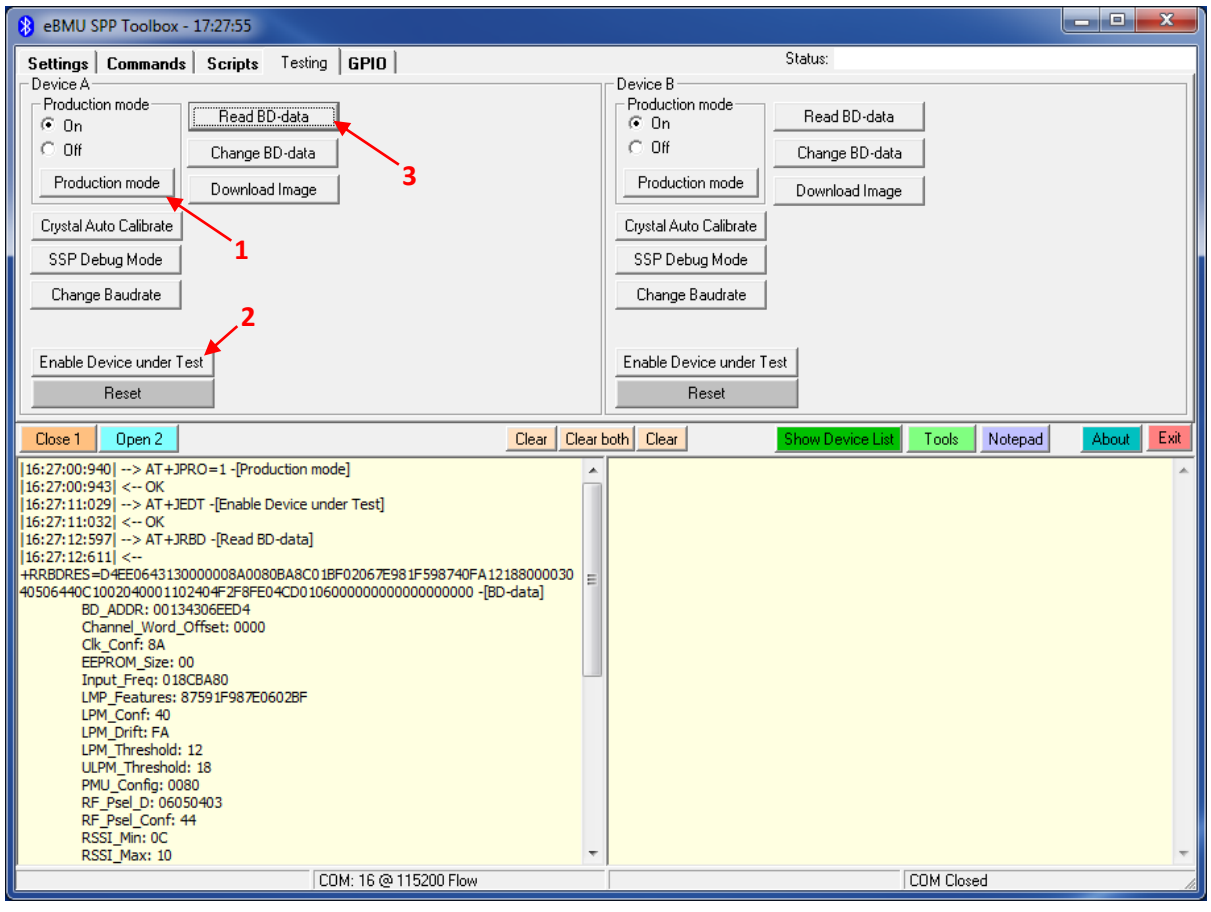


Figure 21. Setup of Device Under Test with AT commands.

When the CBT is connected to eUniStone, all control of the RF measurements is done by the CBT over the air.

9.3 Preparation for RF Tests in non-signaling mode

To perform RF tests in non-signaling mode, i.e. without a BT tester, the eUniStone has to be configured through HCI commands, which can only be executed on the eUniStone HCI application. To use the HCI commands, the HCI application must be downloaded into the EEPROM of the eUniStone module.

Note: Once the HCI application has been loaded to the module, it cannot be reprogrammed over the UART anymore. An I2C programmer is needed to restore the module with the standard SPP application.

The HCI application can be downloaded in 2 ways.

1. Via UART interface only:
    a. Open eBMU SPP Toolbox and connect to device.
    b. Click “Testing” panel
    c. Click the button ”Read BD-data”
    d. Make a note of the BD_ADDR and Osc_Trim (These will be needed later)
    e. Switch to production mode on, by clicking “Production mode”.
    f. Click “Download Image”.
    g. Select image of HCI application and click “Download” (See Figure 22).

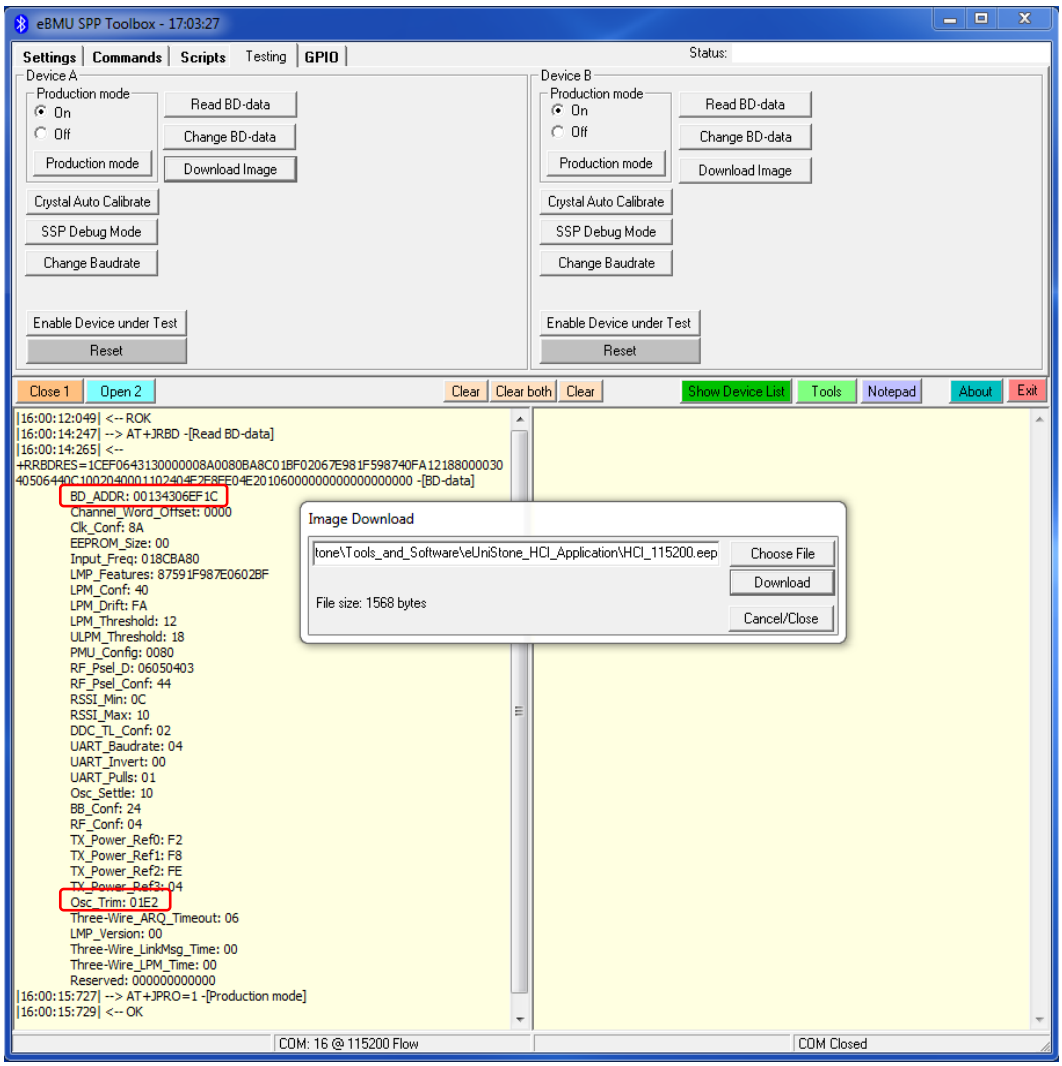


Figure 22. Download of HCI Application via UART

2. Via I2C interface:

Read the BD-data using SPP Toolbox (over UART) and make a note of the BD_ADDR and Osc_Trim. Connect the AARDVARK Programmer and follow the instructions in [4].

When the HCI application is downloaded into EEPROM, issue a HW reset to load the HCI application in eUniStone.

Note: It is important to read out and note the OSC_Trim value before download of the HCI application. Since the download procedure will overwrite the calibrated OSC_Trim value, the original OSC_Trim must be written back into the EEPROM after the download.

The HCI Lite tool provided by Intel allows the configuration for RF transmission tests on the module.

After the RF tests are performed, the SPP-AT application must be downloaded into EEPROM again to enable the usage of eBMU SPP Toolbox. Since the SPP Toolbox cannot work with the HCI application, the SPP-AT application is normally downloaded via I2C interface. This requires a specific I2C programmer. Please follow the instructions in [4]. Restoring of BD-address and OSC_Trim shall be performed after downloading the SPP-AT application. An alternative way of restoring the SPP-AT application is to overwrite parts of the EEPROM. This can be done using HCI Lite as described in the following chapter, 9.4.

9.4 Restoring from HCI to SPP (without using Aardvark)

Restoring from HCI application to SPP-AT application can be done in a few steps without using Aardvark. The complete sequence consists of three major steps.

1. Overwriting the EEPROM using HCI Lite, described in chapter 9.4.1.
2. Loading latest SPP SW version using SPP Toolbox, described in chapter 9.4.2.
3. Writing back BD address and oscillator trim value using SPP Toolbox, described in chapter 9.4.3.

9.4.1 Restoring SPP-AT - Overwriting EEPROM

To overwrite the EEPROM, two commands need to be sent, Intel_Manufacturer_Mode and Intel_Raw_Write_Ext_EEPROM. Both these commands shall be sent using ”Send custom command” found on the ”Connection Setup” tab in HCI Lite, see Figure 23. After overwriting the EEPROM a HW reset shall be performed. The module will then restart. After restart when it tries to read the EEPROM the overwritten part will be missing and the module will instead of loading the HCI SW load and use the default SPP-AT application.

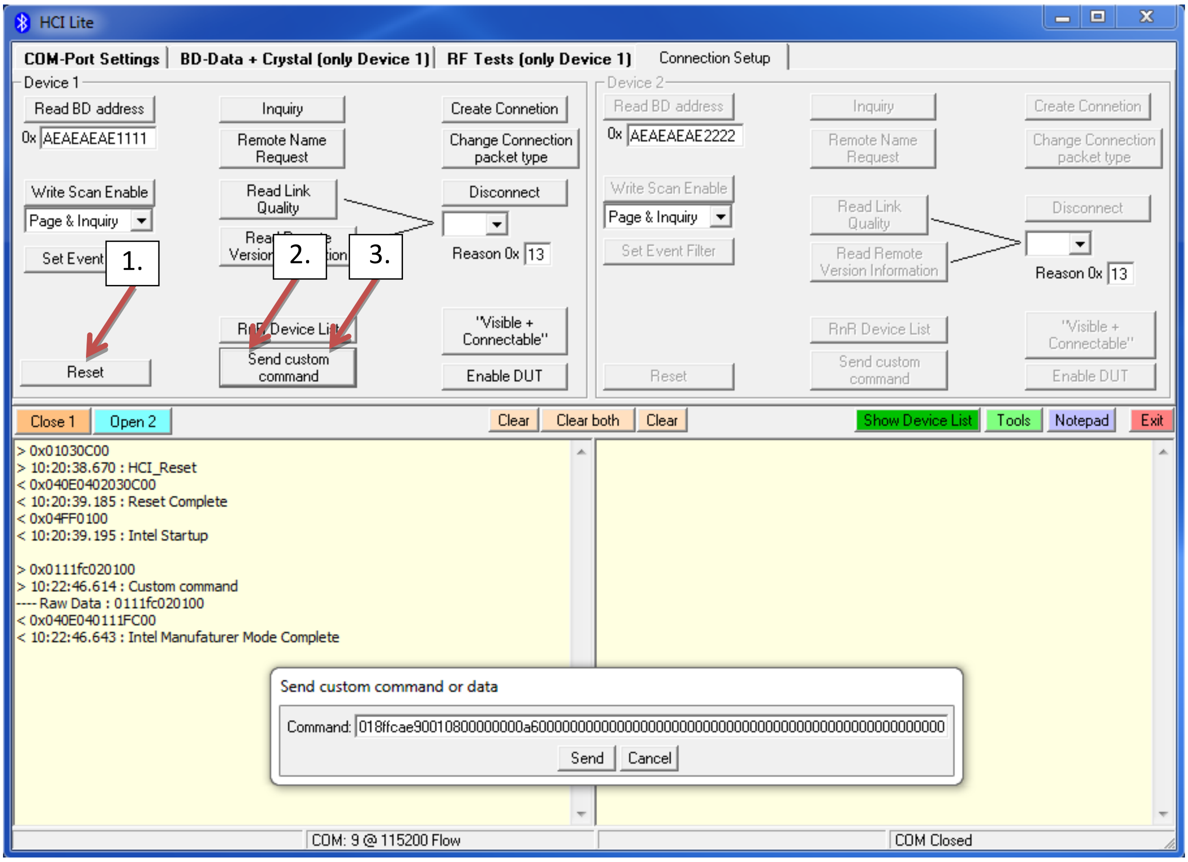


Figure 23 Restoring from HCI to SPP-AT using HCI Lite.

Sequence:

1. Send a HCI reset by clicking the "Reset" button to make sure that the chip/module is in a known state.
2. Click on "Send custom command" and paste 0111fc020100 in the command field, then click "Send"
3. Once more click on "Send custom command", clear the command field and paste
018ffcae90010800000000a600000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000

   Click "Send"

4. Make a HW reset of the board.

After the HW reset, the module will start up running the SPP-AT application, close HCI Lite and start the SPP Toolbox.

9.4.2 Restoring SPP-AT - Loading latest SPP-AT Application.

After starting SPP Toolbox, make another HW reset. “ROK” shall be returned and shown in the log window. When the SW version is read with “Read Revision Information” you will notice that the SW version is 3.0 which is the default SW in the ROM of the chip.

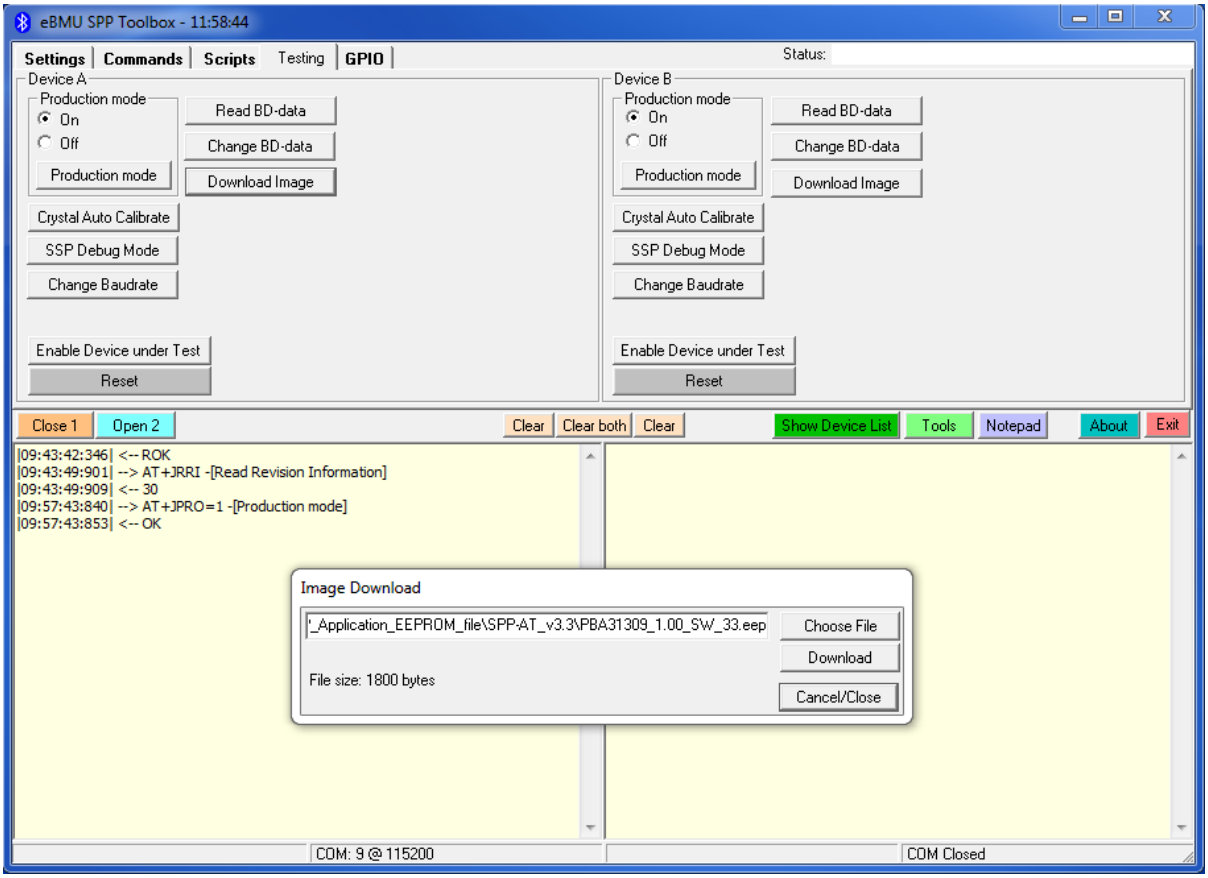


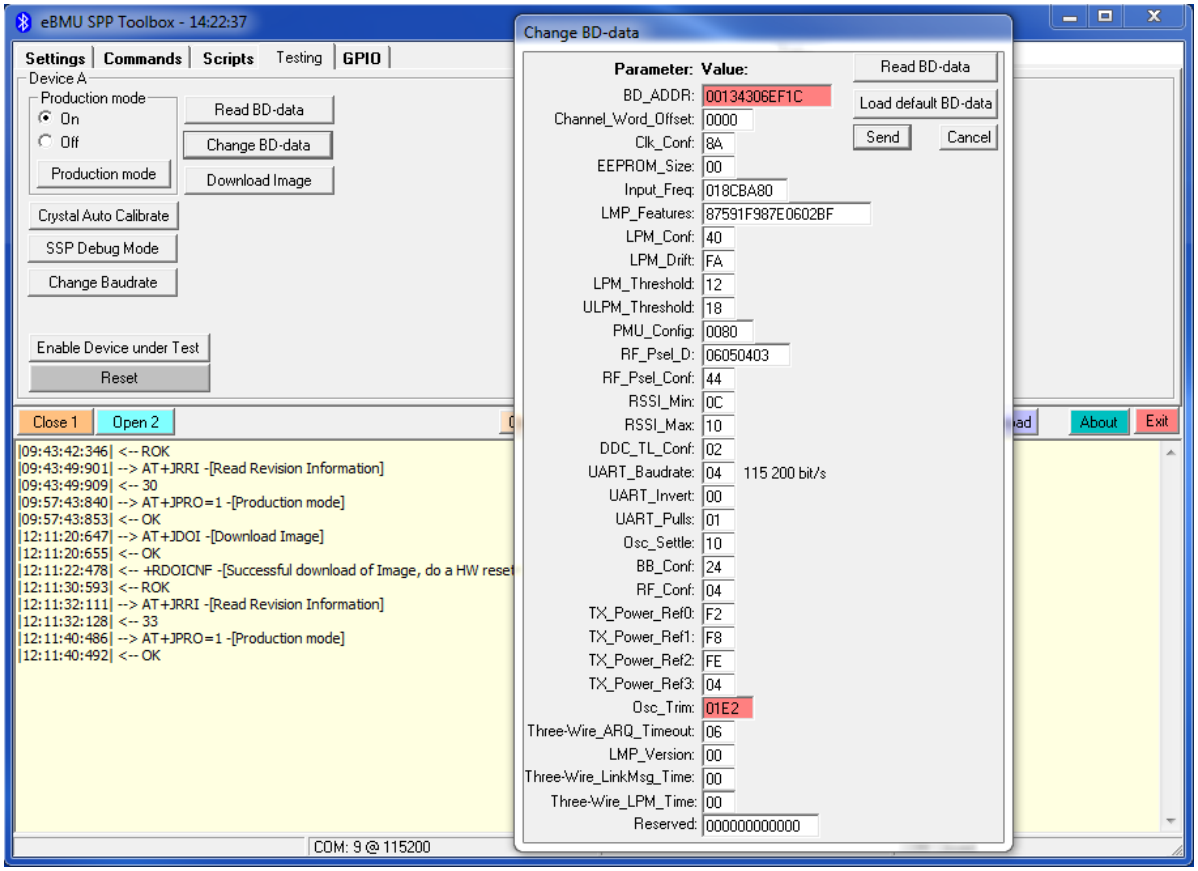
Figure 24 Downloading latest SPP-AT application.

Sequence:

1. Start SPP Toolbox, open the COM port and make a HW reset, ”ROK” is returned.
2. Read SW version. ”30” is returned
3. Download the latest SPP-AT application by first clicking ”Production mode” and then ”Download Image”
4. Choose the latest available SPP-AT application file and click ”Download”
5. After downloading the SPP-AT application, make a HW reset, “ROK” is returned.

9.4.3 Restoring SPP - Writing BD address and oscillator trim value

After loading the latest SPP-AT application, the BD address and oscillator trim values need to be restored.



Sequence continuing from previous chapter:

6. Check that the intended SPP-AT application is used by “Read Revision Information”.
7. Enable production mode by clicking “Production mode”.
8. Fill in BD address (BD_ADDR) and oscillator trim value (Osc_Trim) as in chapter 9.3.
9. Click the “Send” button to write the values.
10. Make a HW reset and make sure that “ROK” is returned.

At this point the full restore is performed. You can check that the correct SPP-AT application version, BD address and oscillator trim values are stored in the module by reading them with the commands “Read BD-data” and “Read Revision Information”.

9.5 Crystal Trimming

The parameter Osc_Trim in BD_Data is calibrated during production in factory. It is used to tune the crystal on the module at the right frequency. It makes use of parallel capacitors switched in and out in order to decrease or increase the clock frequency.

A wrong Osc_Trim value cause problem in

- UART communication with the host
- Shifted TX frequency, which at the end can cause the failure of TX frequency accuracy test (TRM/CA/08/C initial carrier frequency tolerance).

If the calibrated OSC_Trim value has been lost, the internal crystal must be calibrated again before performing any RF tests with the module or putting the module into any real application.

9.5.1 Osc_Trim Parameter

The Osc_trim parameter in the BD_Data is 10 Bits long and the bits 0 to 5 switch binary weighted capacitances from 1xLSB to 32xLSB, where LSB is 40fF. The bits 6 to 9 all have the same value of 2.56 pF each, which is 64xLSB each.

The following table shows the capacitance switched by each bit and how to get the value in hexadecimal notation.

Table 11. Register for Switching Capacitances

2560	2560	2560	2560	1280	640	320	160	80	40	Cap in fF
9	8	7	6	5	4	3	2	1	0	Bit
0		0				0				= 0 pF
1		9				F				= 6.36 pF
3		F				F				= 12.76 pF

There are 10 internal capacitances between the pins LOAD and VSS and every bit of the array represents one of the capacitances. All of them are switchable so it is possible control whether they are connected or not. With setting a bit to 1 the capacitance is connected.

Table 11 shows 3 examples of a value for the Osc_Trim and the real capacitance. The first example is the minimum, the second the proposed start value for the tuning algorithm and the last the maximum achievable value. The frequency gets lower with a higher value of the capacitance array.

9.5.2 Crystal Trimming Procedure

A 32 MHz clock is derived internally from the crystal oscillator. It can be switched to the GPIO pin P0.1 or P0.8.

The reference signal must be adjusted with a precision that is determined by the total acceptable deviation of ±20 ppm for the Bluetooth reference clock. The sum of adjustment precision, variation over temperature and ageing must remain within this range. The best achievable precision for the reference clock adjustment is ±2 ppm (±64 Hz) for PBA31309.

A frequency counter is needed with a sufficient precision, for example the Agilent 53131A Universal counter.

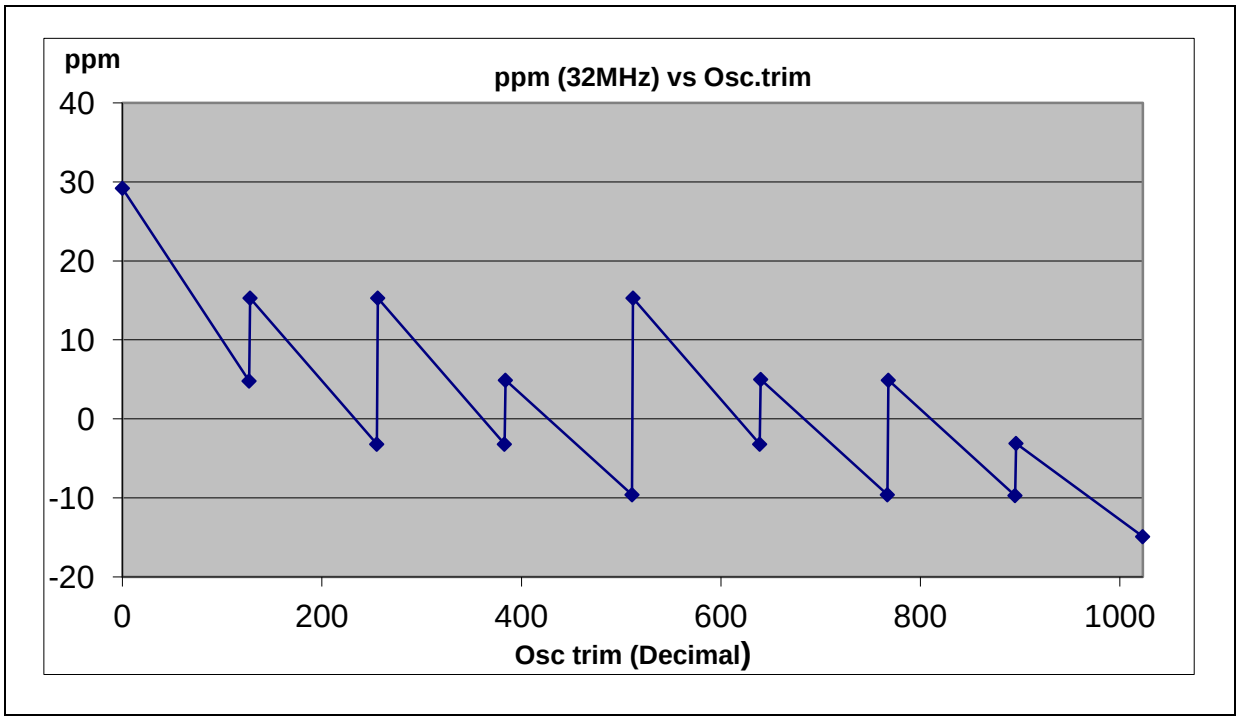


Figure 25. Typical 32MHz frequency offset in PPM versus OSC_Trim value.

9.5.2.1 Crystal Trimming with HCI application for special RF tests

After the download of the HCI application, the calibrated OSC_Trim value is overwritten with the default value. If the calibrated value has not been noted before the download, an additional crystal trimming must be performed before any RF tests.

With HCI Lite the crystal trimming can be performed easily. The 32 MHz clock signal can be switched to GPIO pin P0.1 or P0.8 using the HCI Lite tool (see in Figure 26). Then the clock frequency can be measured by a frequency counter connected to the test pin.

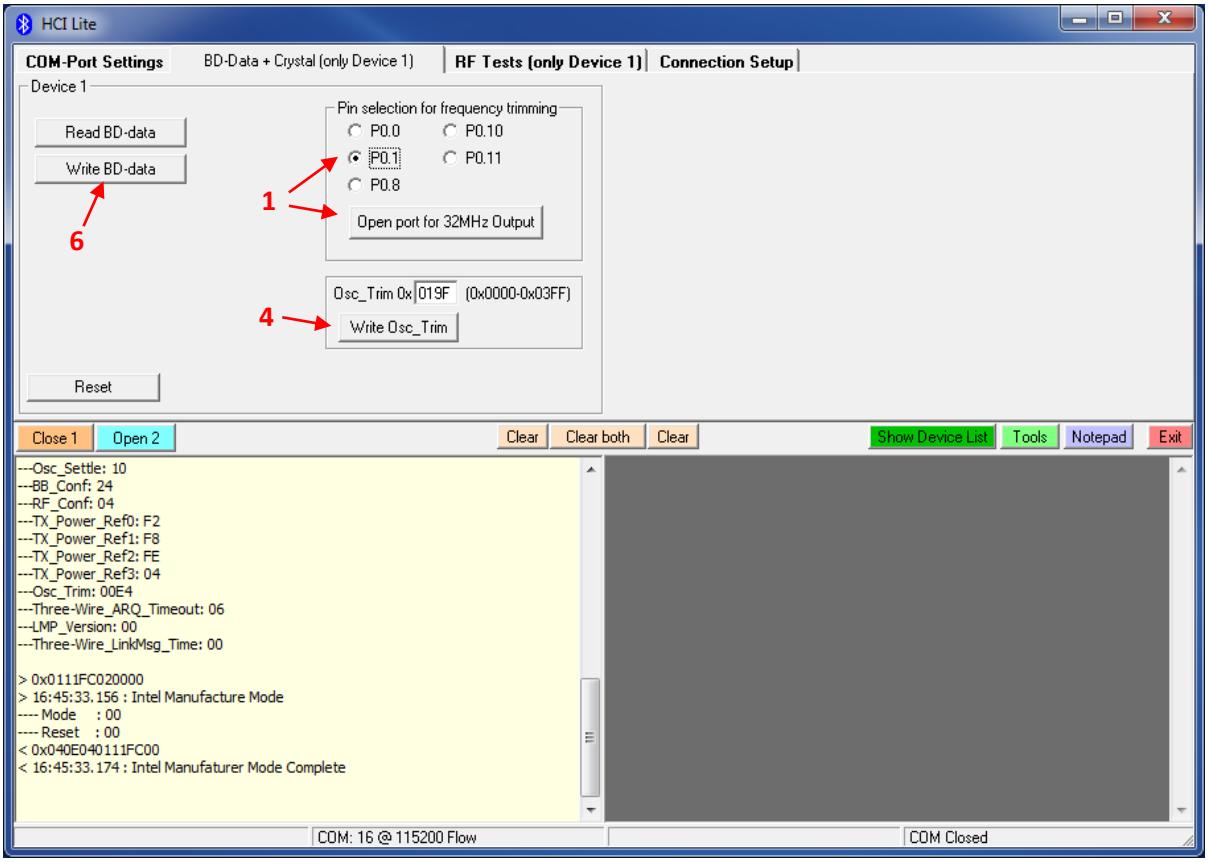


Figure 26. Crystal Trimming using HCI Lite Tool

The crystal trimming procedure using a frequency counter and the HCI Lite tool is as following:

1. Make the 32 MHz clock available at P0.1 or P0.8 using the HCI_Lite tool. (See in Figure 26)
   a. Select P0.1 or P0.8 pin for output of the 32 MHz clock
   b. Click “Open port for 32 MHz”
2. Connect the frequency counter to the appropriate test pin P0.1 or P0.8.
3. The 32 MHz clock signal can now be measured by the frequency counter.
4. Write the trimming value to the Osc_Trim register through the HCI Lite tool. The start value is 0x19F. (See in Figure 26)
   a. Enter the Osc_Trim value in the text box for Osc_Trim
   b. Click “Write Osc_Trim”

5. Now the frequency has to be adjusted by repeating step 4 with different values. The frequency will be lower with a higher value of the capacitance array.

6. When the desired accuracy of the 32 MHz clock is obtained, store the corresponding trim value to the parameter Osc_Trim of the BD-data. This can be done by clicking the button “Write BD-Data” in HCI Lite tool. In the pop-up window, only change the parameter OSC_Trim to the calibrated value and give the module an individual BD_ADDR. (See in Figure 27)

7. Issue a HW reset of the device.

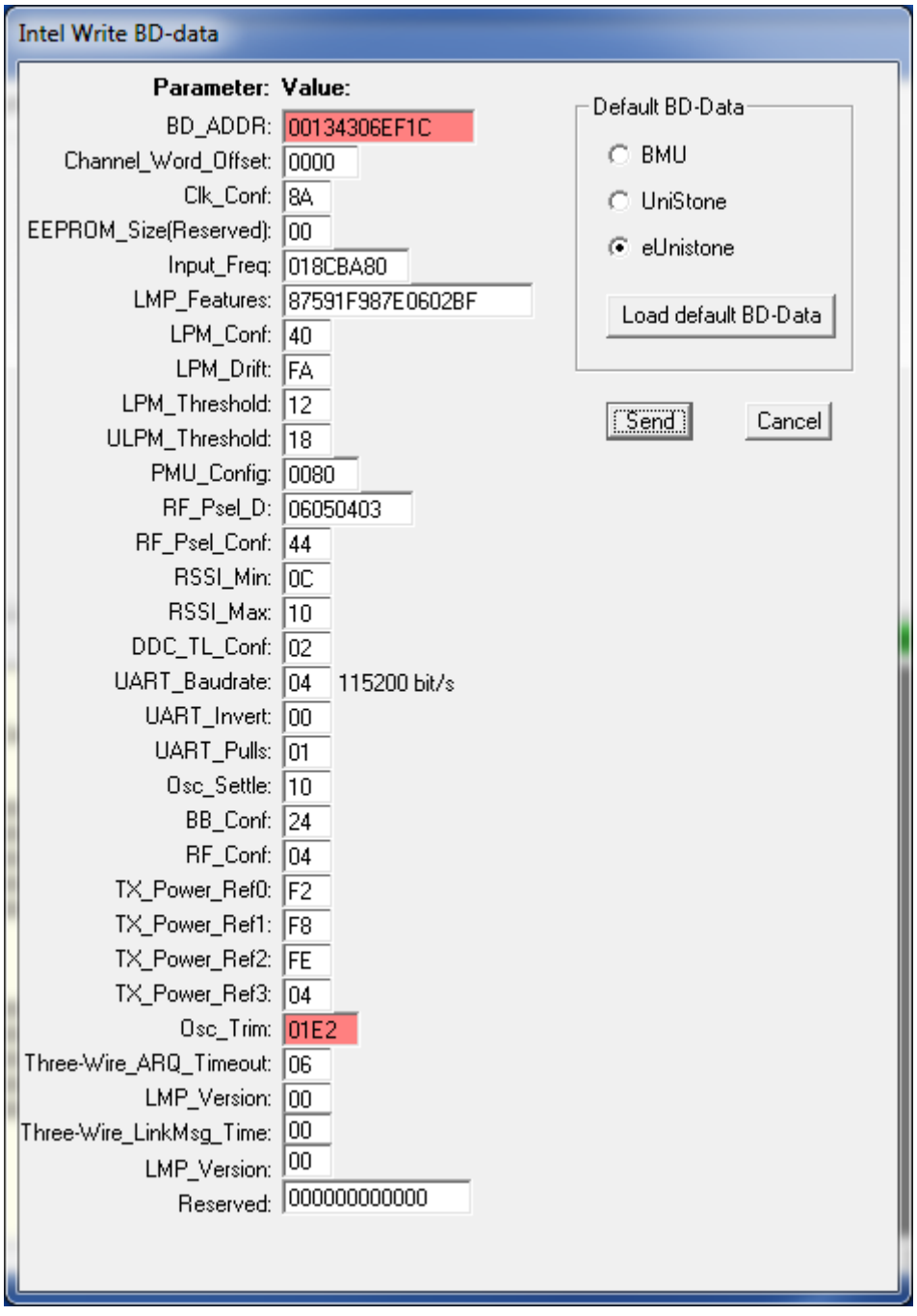


Figure 27. Intel Write BD-Data Window

9.5.2.2 Crystal Trimming with SPP-AT Application

When the SPP-AT application is loaded into the EEPROM, the crystal trimming can be performed via AT commands using the eBMU SPP Toolbox provided by Intel.

This is only necessary if the original calibration value, which is programmed during module production at Intel, has been lost.

1. Connect a frequency counter to the test point P0.1 or P0.8.
2. Enable the production mode using the command *AT+JPRO=1*.
3. Issue the following command that makes the internal reference clock available at the test point (32 MHz generated by the 26 MHz crystal oscillator):

   AT+JCAC = <osc_trim_value>,<GPIO>

   Where:

   a. <Osc_trim_value>: Range from 0x000 to 0x3FF

   b. <GPIO>:
   0x0002 to output 32 MHz on pin P0.1
   0x0100 to output 32 MHz on pin P0.8
4. Measure the frequency of the signal on pin P0.1/P0.8 with the counter.
5. Trim the 32 MHz frequency to be within ±2ppm (±64 Hz) of accuracy changing the field *<osc_trim_value>*.
6. When the desired accuracy is obtained, write the corresponding trim value to the parameter Osc_Trim in the BD_Data using the command *AT+JCBD=<bd_data>*.
7. Issue a HW reset of the device.

The SPP Toolbox provides a friendly user interface which enables the user to execute the AT commands simply by clicking several buttons (see in Figure 28).

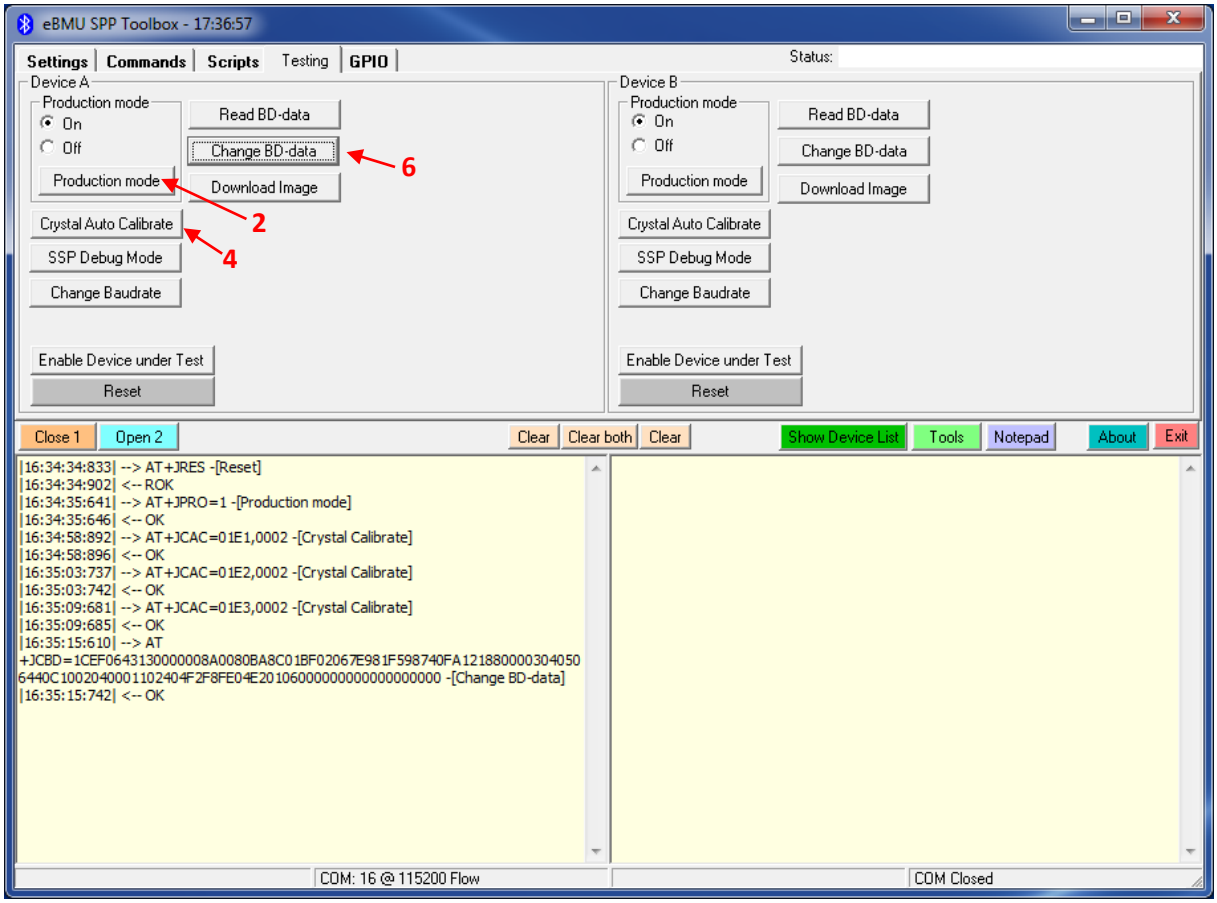
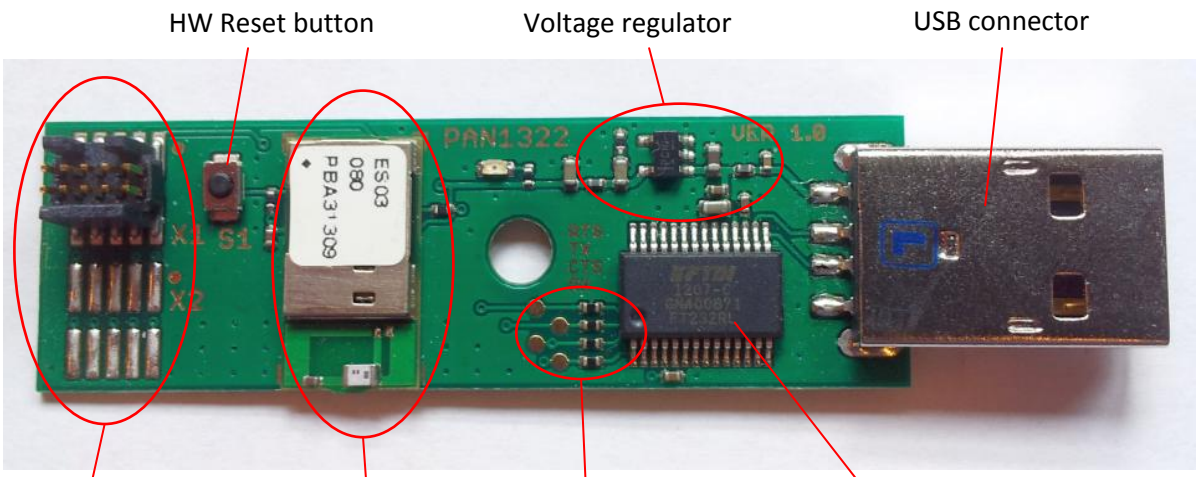


Figure 28. Crystal Trimming using eBMU SPP Toolbox

10 FAQ

10.1 First use of PBA31309 USB dongles

The USB dongle for PBA31309 contains the eUniStone module itself, an FTDI chip as interface and a voltage regulator that use the USB connectors 5 volt to power the complete dongle. The FTDI chip work as UART over USB interface. This means that when the dongle is connected to the computer a port, looking like an ordinary COM port will be available in the computer.



10.1.1 Connect the dongle to a USB port on the computer.

The computer will automatically start to install the drivers for the FTDI chip. If you have had any other device connected that use FTDI, the installation should be finalized directly and a new COM port should be available in the device manager. If you have not had any FTDI device connected to the computer before it might take some time for the computer to find the driver. The computer might also search Microsoft driver data base on internet. If Windows is not able to find any driver you need to download it from FTDI, http://www.ftdichip.com/Drivers/VCP.htm After connecting the USB dongle you should be able to find it in the Device Manager as in the below picture.

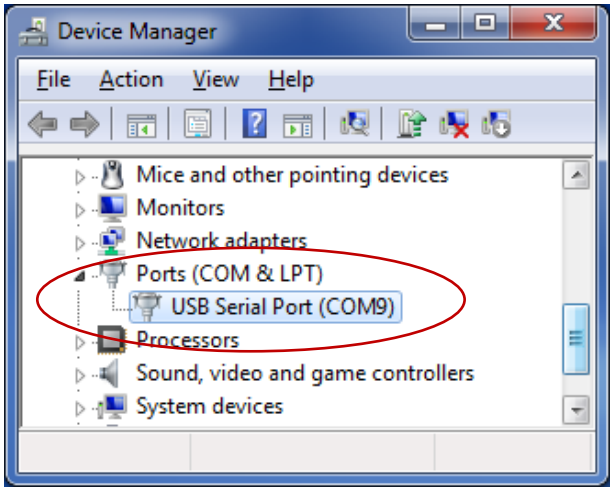


Figure 29. Finding the COM port in the Device Manager

10.1.2 Modify FTDI COM port settings for full throughput.

The Latency Timer parameter of the FTDI driver is set to 16ms as default after installing the FTDI driver. To get full throughput with higher baud rates than 115200 bit/s this parameter need to be changed to 1ms. This is done by right-clicking on the COM port in the device manager and choosing "Properties". See the below sequence and Figure 30.

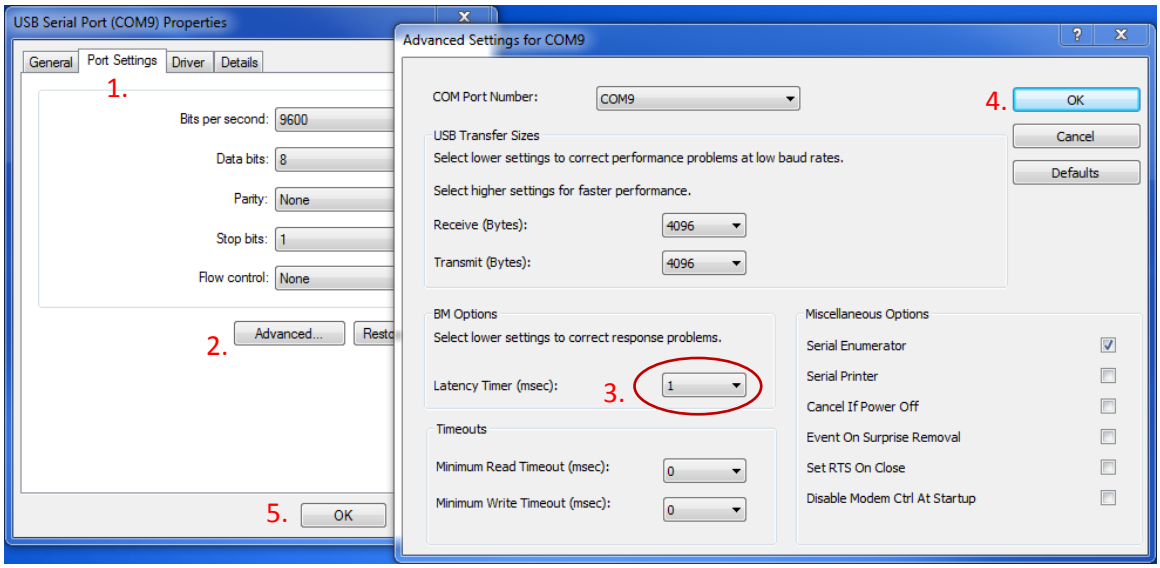


Figure 30. Modify the Latency Timer of the FTDI driver

1. On the properties dialogue, click on tab "Port Settings".

2. Click on “Advanced”.
3. In the Advance Settings dialogue, change the Latency Timer to 1ms.
4. Click the button “Ok” on the Advance Settings dialogue.
5. Finally, click the “Ok” button on the properties dialogue.

Observe that this needs to be done for each new virtual FTDI COM port. E.g. for each connected USB dongle.

10.2 Change UART baud rate of PBA31309

The default baud rate of the PBA31309 is 115200 bit/s. If another baud rate is needed it can be changed by the customer. Changing the UART baud rate can be performed in two ways. Either the new baud rate is stored in the EEPROM of the module during product manufacturing or the baud rate is temporary changed by the host in runtime.

10.2.1 Change of baud rate in product manufacturing

Step	Command/response	Description
1	→ AT+JRES	Send software Reset to the module
	← ROK	Make sure the module respond with ROK
2	→ AT+JRBD	Read BD-data
	← +RRBDRES=083C0E19030000008A0080BA8C01BF02067E981F598740FA12188000030405064 40C100**04**0001102404F2F8FE04B900060000000000000000000	The module will respond with the complete BD-data. (UART baud rate is marked with bold).
3	→ AT+JPRO=1	Enable production mode
	← OK	The modules shall response with OK
4	→ +JCBD=083C0E19030000008A0080BA8C01BF02067E981F598740FA12188000030405064 40C100**07**0001102404F2F8FE04B900060000000000000000000	Write the BD-data with the Change BD-data command. In this example 0x**07,** which will set the default baud rate to 921699 bit/s.
	← OK	The modules shall response with OK
5	→ AT+JRES	Send software Reset to the module to make it use the written BD-data and new baud rate. Observe that that the response from the module will be sent on the new baud rate.
	CHANGE UART BAUD RATE OF HOST	
	→ AT+JRES	Send software Reset from the host on the new baud rate
	← ROK	Make sure the module respond with ROK.

10.2.2 Change of baud rate by host at runtime

Changing the baud rate at runtime is done with the command AT+JSBR=<baud_rate>. This baud rate will be used by the module until it's changed again. A HW or SW reset will make the module use the baud rate stored in EEPROM, (default 115200 bit/s)

Table 12. Changing the baud rate at runtime

Step	Command/response	Description
1	→ AT+JSBR=0921600	Change baud rate to 921600 bit/s
	← OK	An OK will indicate that the baud rate has changed. (The OK is sent on the old baud rate)
2	CHANGE BAUD RATE ON HOST	
3	Continue sending commands and/or data on the new baud rate	

10.3 SW update

The software update of PBA31309, has to be done in a certain way to make sure that the correct Bluetooth address (BD_ADDR) and the oscillator trim (OSC_Trim) values does not get lost in the process. The software update can be performed by either the SPP Toolbox or with a customer specific tool or application. In the below table you will find the needed commands and the order of them.

Table 13. SW update sequence

Step	Command/response	Description
1	→ AT+JRES	Send software Reset to the module
	← ROK	Make sure the module respond with ROK
2	→ AT+JRBD	Read BD-data
	← +RRBDRES=**083C0E190300**00008A0080BA8C01BF02067E981F598740FA12188000030405064 40C1002040001102404F2F8FE04**B900**06000000000000000000	The module will respond with the complete BD-data including the Bluetooth address and oscillator trim value, (marked with bold). Observe that the values are in reverse byte order. The real address in this example is 0x0003190E3C08 and the oscillator value is 0x00B9
3	→ AT+JPRO=1	Enable production mode
	← OK	The modules shall response with OK
4	→ AT+JDOI	Enable software/patch download
	← OK	The modules shall response with OK
5	DOWNLOAD THE SOFTWARE	The software shall be downloaded by binary writing one byte at the time until the complete SW file has been loaded to the module.

Step	Command/response	Description
	← +RDOICNF	The module shall indicate with an event that the software/patch has been successfully downloaded
6	MAKE A HW RESET	To enable the new software a hardware reset is needed
	← ROK	The module has now restarted and runs the new software, but with default Bluetooth address (0003199E8B71) and default oscillator trim value (0218).
7	→ AT+JPRO=1	Enable production mode
	← OK	The modules shall response with OK
8	→ +JCBD=**083C0E190300**00008A0080BA8C01BF02067E981F598740FA12188000030405064 40C10020400011024 04F2F8FE04**B900**06000000000000000000	Write BD-data with the address and oscillator trim value read out in step 2.
	← OK	The modules should response with OK
9	→ AT+JRES	Send software Reset to the module to make it use the written BD-data.
	← ROK	Make sure the module respond with ROK.

10.4 Aardvark and eeprog.exe

Aardvark is an I2C/SPI host adapter from Total Phase. Aardvark can be used to update or restore software in the EEPROM of PBA31309. Eeprog.exe and the Aardvark drivers are available on the development CD. Eeprog.exe is special DOS software made for downloading and reading EEPROM data. More information about Aardvark can be found here:
http://www.totalphase.com/products/aardvark_i2cspi/

10.4.1 Installing Aardvark

Connect Aardvark to the USB port of the computer.

When the driver installation started choose to install the driver manually. The driver is found on the development CD.

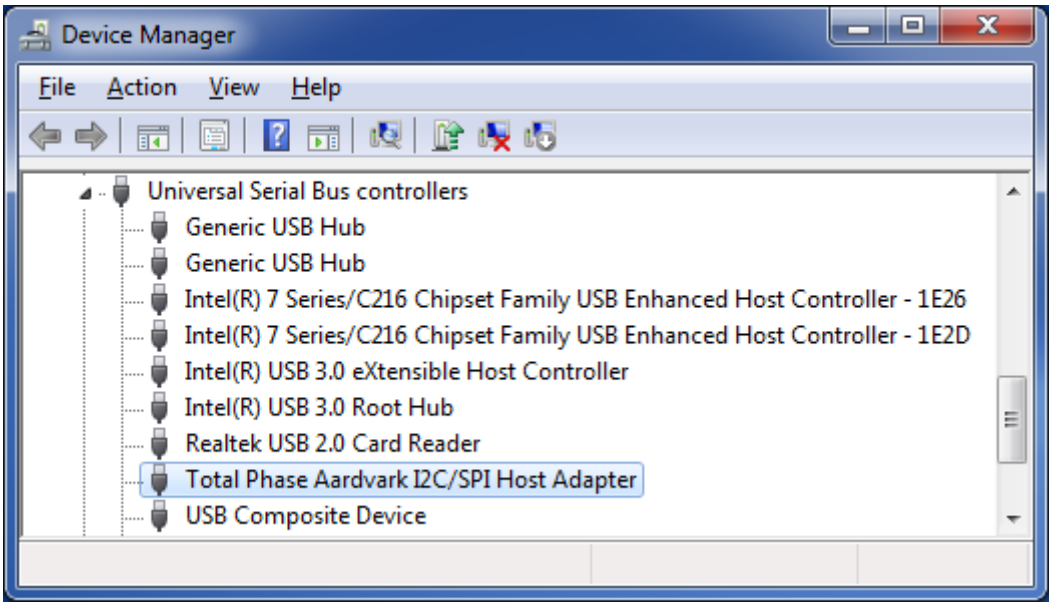


Figure 31. Aardvark drivers installed and visible in the Device Manager.

10.4.2 Connecting Aardvark to PBA31309

Aardvark shall be connected to eUniStone with the signals I2C clock, I2C data and ground. See Figure 32 for detail on how this is done with eUniStone on the USB dongle.

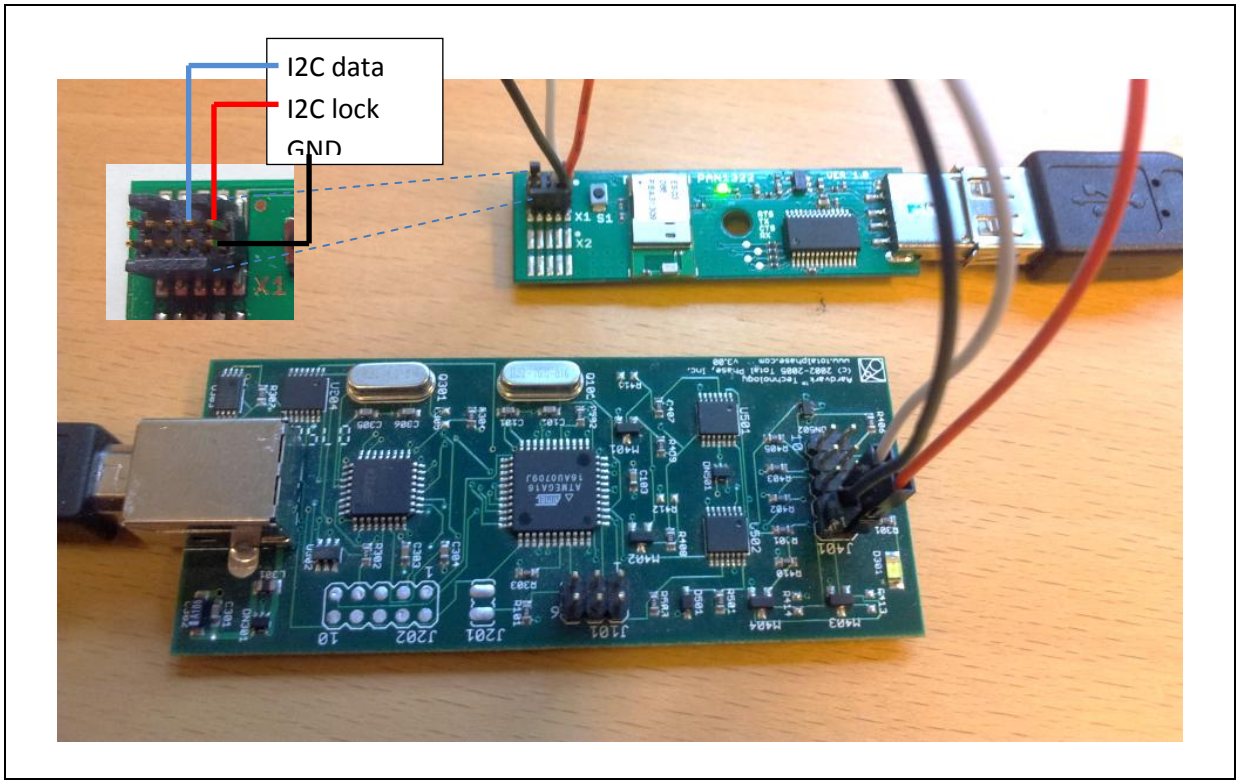


Figure 32. Aardvark connect to the PBA31309 USB dongle. (Black = GND, Red = I2C clock, White = I2C data)

10.4.3 Downloading software to PBA31309

To use eeprox.exe for downloading software the following syntax is used:

C:\Aardvark>eeprog.exe –aa –w 0 binary_file.eep

-aa	Use Aardvark I2C host adapter to access EEPROM
-w	Write to EEPROM
0	Address in the EEPROM where Aardvark shall start to write.
Binary_file.eep	The file that shall be written

10.4.4 Aardvark problems.

The Aardvark device use a similar FTDI chip as is used on the PBA31309 USB dongles. In some cases when the eeprog.exe tries to access the Aardvark device, it cannot connect to it and an error message is shown in the DOS window as below

C:\Aardvark>eeprog.exe –aa –w 0 binary_file.eep

Error: Failed to open Aardvark device on port 0

This can be overcome by powering the PBA31309 USB dongle with a USB hub

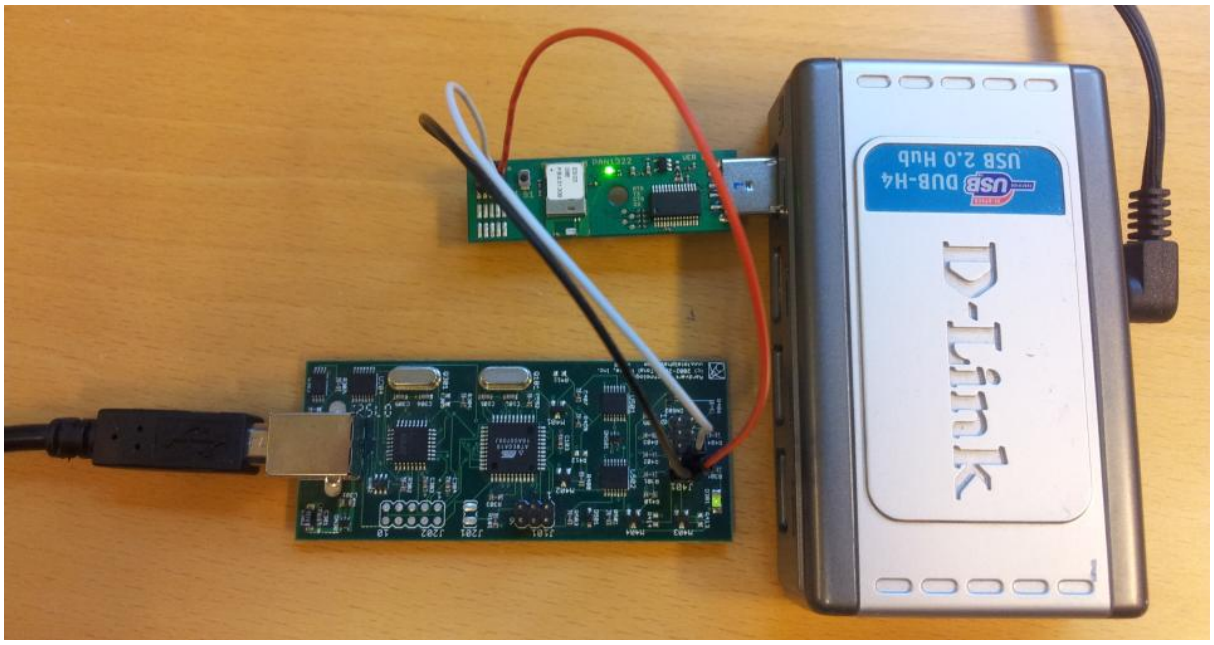


Figure 33. PBA31309 powered by an USB hub

Observe that the USB hub shall not be connected to the computer that Aardvark is connected to. Also does the USB hub need to be powered so that it can power PBA31309.

10.5 UUID & CoD

Please see chapter 5.1.3.3 for general details about UUID and Class of Device, CoD. See chapter 5.2 for details about Android, iPhone and Windows Phone 8.

10.6 Low Power Mode, LPM, control

Please see chapter 2.1.1 for details about LPM control of eUniStone.

10.7 Bluetooth Qualification and Regulatory Certification

Please see chapter 8 in document ” eUniStone_V1.0_UM_HD_Rev1.1.pdf” for details about BTQ and Regulatory Certification.

11 References

No./Name	Title	Source
1	eUniStone Hardware Description	eUniStone_V1.0_UM_HD_Rev1.0.pdf
2	eUniStone SPP-AT specification	eUniStone_V1 00_UM_SD.pdf
3	eUniStone SW release notes	eUniStone_V1.00_SW_3.1.pdf
4	Instructions on how to download software using Aardvark	Instructions_EEPROM_download.pdf
5	Aardvark I2C/SPI Host Adapter	http://www.totalphase.com/products/aardvark_i2cspi
6	Bluetooth Accessory Design Guidelines for Apple Products R6	https://developer.apple.com/hardwaredrivers/BluetoothDesignGuidelines.pdf
7	Mfi Accessory Firmware Specification R46	Available under Mfi license at https://mfi.apple.com
8	Mfi Accessory Hardware Specification R9	Available under Mfi license at https://mfi.apple.com
9	iPod Authentication Coprocessor 2.0C Specification R1	Available under Mfi license at https://mfi.apple.com
10	Qt Project	http://qt-project.org
11	Eclipse	http://www.eclipse.org
12	Android Developers	http://developer.android.com
13	Apple iOS developer	https://developer.apple.com/devcenter/ios/
14	Windows Phone 8 developer	http://developer.windowsphone.com/en-us
15	eUniStone Demo Specification	eUniStone_V1.00_3D_Demo_Specification_Rev1.0.pdf