Category:Bluetooth

Contents

Stable: 17.06.2020 - 15:20 / Revision: 10.01.2020 - 15:21

A quality version of this page, approved on *17 June 2020*, was based off this revision.

This category groups together all articles related to the Linux®**Bluetooth** software framework.

It is recommended to first read the Bluetooth overview article.

Linux® is a registered trademark of Linus Torvalds.

Pages in category "Bluetooth"

The following 5 pages are in this category, out of 5 total.

- Bluetooth device tree configuration
- Bluetooth overview
- How to scan BLE devices
- How to scan Bluetooth devices
- How to set up a Bluetooth connection

Stable: 11.06.2020 - 09:33 / Revision: 11.06.2020 - 06:41

A quality version of this page, approved on *11 June 2020*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure *Bluetooth* [1] **when the peripheral *(or peripheral associated to the framework)* is assigned to the Linux$^{®}$OS**.

The configuration is performed using the **device tree mechanism** [2].

The Bluetooth companion chip chosen on our platform is a Cypress chip [3]

2 Bluetooth DT bindings documentation

The *Bluetooth*[4] **tree bindings are composed of:**

- STM32 USART device tree bindings [5]
- The Cypress device, used as child node [6] of the host USART device to which the slave device is attached.

3 Bluetooth DT configuration

This hardware description is a combination of the STM32 microprocessor device tree files (.dtsi extension) and board device tree files (.dts extension). See the device tree for an explanation of the device tree file split.

3.1 Bluetooth DT configuration (STM32 level)

The USART peripheral node is located in *stm32mp151.dtsi*

- This is a set of properties that may not vary for given STM32 device, such as: registers address, clock, reset...

The USART DT configuration is explained in Serial TTY device tree configuration

3.2 Bluetooth DT configuration (board level)

```
&usart2 {
	...
	uart-has-rtscts;                                     /* enable hardware flow
control */
	...
	bluetooth {                                           /* node of Bluetooth
companion chip */
		shutdown-gpios = <&gpioz 6 GPIO_ACTIVE_HIGH>;  /* GPIO specifier, used to
enable the BT module */
		compatible = "brcm,bcm43438-bt";
		max-speed = <3000000>;
	};
};
```

Specific properties for USART:

- uart-has-rtscts; bool flag to enable hardware flow control

4 How to configure Bluetooth using CubeMX

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.
The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.

5 References

- Bluetooth
- Device tree
- MURATA CYW4343W datasheet
- WLAN_and_Bluetooth_hardware_component
- Serial TTY device tree configuration
- Documentation/devicetree/bindings/net/broadcom-bluetooth.txt

Linux® is a registered trademark of Linus Torvalds.

Operating System

Device Tree

Universal Synchronous/Asynchronous Receiver/Transmitter

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

BlueTooth

Stable: 15.04.2020 - 08:31 / Revision: 15.04.2020 - 08:29

A quality version of this page, approved on *15 April 2020*, was based off this revision.

This article explains how a Linux® Bluetooth framework is composed, how to configure it, and how to use it.

Contents

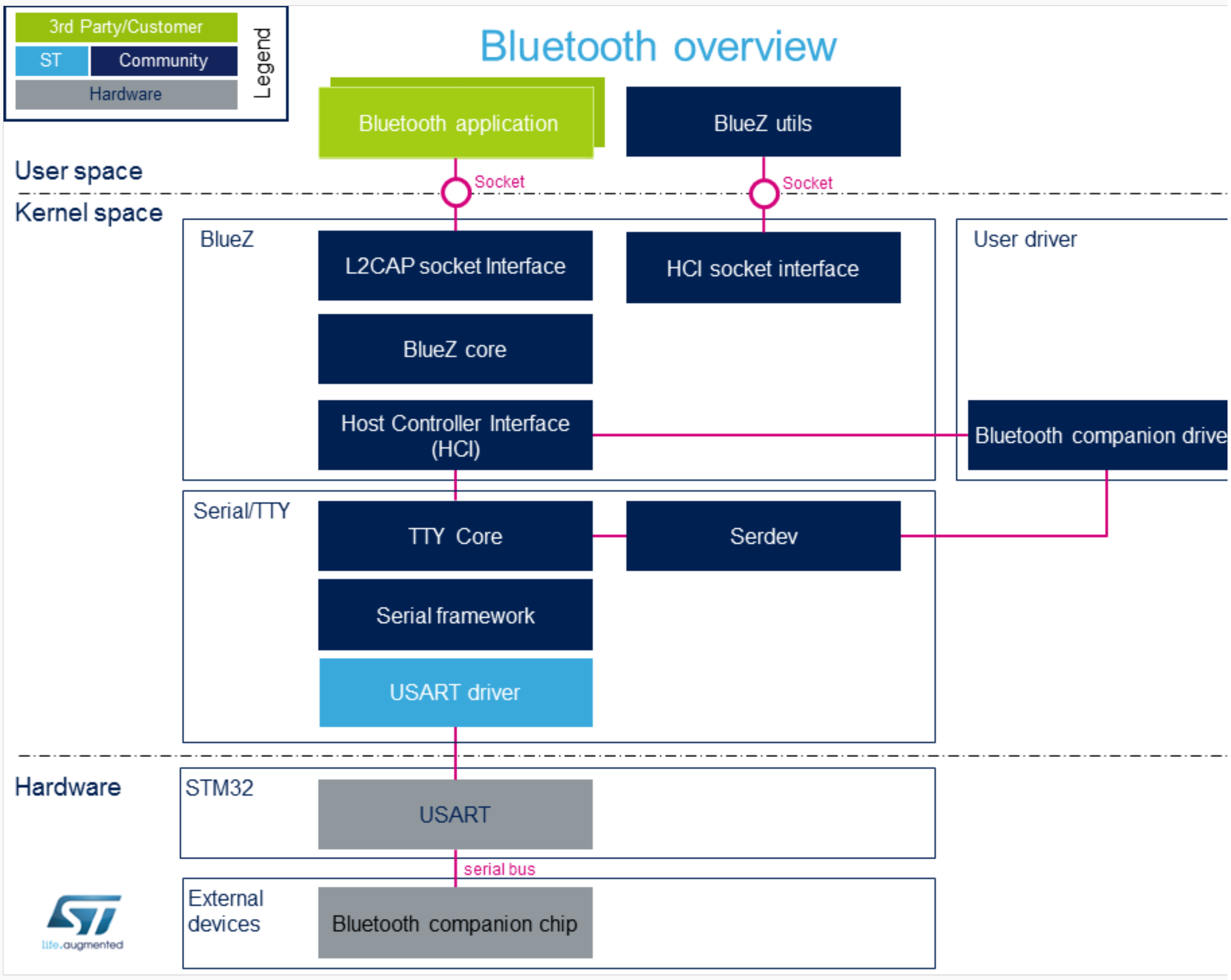
1 Framework purpose

Bluetooth is a protocol for wireless communication over short distances. It was developed in the 1990s to reduce the need for cable interconnects. Devices such as mobile phones, laptops, PCs, printers, digital cameras and video game consoles can connect to each other and exchange information using radio waves. This can be done securely. Bluetooth is only used for relatively short distances, typically of a few meters.
The Linux kernel has a popular Bluetooth stack: BlueZ. This stack is included in most Linux kernels, and runs in both the user space and kernel space of the Bluetooth protocol.
Bluetooth Low Energy is completely supported at the kernel level in Linux.

Bluetooth can be used in many different use cases, as mentioned in the How to use Bluetooth section:

- how to set up a Bluetooth connection Setup Bluetooth
- how to scan Bluetooth devices Scan Bluetooth devices
- how to scan BLE devices Scan BLE devices

2 System overview



Bluetooth Overview

2.1 Component descriptions

From User space to hardware

- **Bluetooth Applications** (User space)

Lots of applications use bluetooth:

bluetoothd [1]: bluetoothd daemon, which manages all the Bluetooth devices

...

- **BlueZ Utils** (User space)

Development and debugging utilities for the bluetooth protocol stack

There is a set of utilities to manage Bluetooth devices:

bluetoothctl [2]: Pairing a device from the shell is one of the simplest and most reliable options

To see all other utilities: https://www.archlinux.org/packages/extra/x86_64/bluez-utils/

- **BlueZ** (Kernel space)

BlueZ [3] is the official Linux Bluetooth stack. It provides, in a modular way, support for the core Bluetooth layers and protocols.
Currently BlueZ consists of many separate modules:
- bluetooth kernel subsystem core
- a "controller stack" containing the timing critical radio interface like HCI [4]
- a "host stack" dealing with high level data like L2CAP [5]

- **Bluetooth companion driver** (Kernel space)

Bluetooth companion driver registers and controls the Bluetooth device

- **Serial/TTY** (Kernel space)

See Serial TTY overview

- **SoC: USART** (Hardware)

See Serial TTY overview

2.2 APIs description

The BlueZ API [6] is documented in the Linux Kernel:
BlueZ exposes a socket API that is similar to network socket programming; the is socket created, used to communicate, PF_BLUETOOTH protocol family [7]

3 Configuration

3.1 Kernel configuration

Bluetooth must be enabled in the kernel configuration, as shown below. On top of this, the user has to activate STM32 support and STM32 USART support. The user can use the Linux Menuconfig tool: Menuconfig or how to configure kernel and select:

```
[*] Networking support  --->
    [*]   Bluetooth subsystem support
        [*]   Bluetooth Classic (BR/EDR)
features
        [*]   Bluetooth High Speed (HS)
features
        [*]   Bluetooth Low Energy (LE) features
        [*]   Bluetooth device drivers   --->
                [*]   HCI UART driver
[*] Device Drivers  --->
    [*]   Character devices  --->
        [*]   Serial device bus
[*] Security options  --->
    [*]   Enable access key retention support
```

For example if the companion chip is the Murata product 1DX[8]

```
[*] Networking support  --->
    [*]   Bluetooth subsystem support  --->
        [*]   Bluetooth device drivers   --->
                [*]   Broadcom protocol support
```

3.2 Device tree

DT bindings documentation deals with all of the required and optional device tree properties.

Detailed DT configuration for STM32 internal peripherals: Bluetooth device tree configuration.

4 How to use Bluetooth

4.1 How to use the Bluetooth user space interface

Please see the examples based on the following use cases:

- how to set up a Bluetooth connection Setup Bluetooth
- how to scan Bluetooth devices Scan Bluetooth devices
- how to scan BLE devices Scan BLE devices

5 How to trace and debug the framework

This part is an example based on the Murata companion chip

5.1 How to verify than Bluetooth driver is correctly probed

- In dmesg log, check "usart" logs :

```
[    0.485894] STM32 USART driver initialized
[    0.487163] 4000e000.serial: ttySTM1 at MMIO 0x4000e000 (irq = 21, base_baud =
4000000) is a stm32-usart
[    0.487514] stm32-usart 4000e000.serial: interrupt mode used for rx (no dma)
[    0.487531] stm32-usart 4000e000.serial: interrupt mode used for tx (no dma)
```

And, if the companion chip is the Murata 1DX :

```
[   13.755069] Bluetooth: HCI device and connection manager initialized
[   13.800349] Bluetooth: HCI socket layer initialized
[   13.837861] Bluetooth: L2CAP socket layer initialized
[   13.843218] Bluetooth: SCO socket layer initialized
[   14.279668] Bluetooth: HCI UART driver ver 2.3
[   14.282780] Bluetooth: HCI UART protocol H4 registered
[   14.288198] Bluetooth: HCI UART protocol Broadcom registered
[   14.289402] hci_uart_bcm serial0-0: No reset resource, using default baud rate
[   14.465008] Bluetooth: hci0: BCM: chip id 94
[   14.469843] Bluetooth: hci0: BCM: features 0x2e
[   14.497113] Bluetooth: hci0: BCM43430A1
[   14.499593] Bluetooth: hci0: BCM43430A1 (001.002.009) build 0000
```

6 References

- [1], bluetoothd
- [2], bluetoothctl
- [3], BlueZ
- [4], HCI
- [5], L2CAP
- [6], BlueZ API
- [7], Socket
- [8], 1DX

Linux® is a registered trademark of Linus Torvalds.

Bluetooth Low Energy. Bluetooth LE, marketed as Bluetooth Smart is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group aimed at novel applications in the healthcare, fitness, beacons, security, and home entertainment industries.
Compared to Classic Bluetooth, Bluetooth Smart is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range. (sourcehttps://en.wikipedia.org/wiki/Bluetooth_Low_Energy)

TeleTYpewriter

Universal Synchronous/Asynchronous Receiver/Transmitter

Application programming interface

High Speed (MIPI® Alliance DSI standard)

Universal Asynchronous Receiver/Transmitter

Device Tree

Stable: 03.02.2020 - 08:42 / Revision: 03.02.2020 - 08:29

A quality version of this page, approved on *3 February 2020*, was based off this revision.

This page lists the different operations needed to scan, connect and display BLE device information. BLE stands for **B**luetooth **L**ow **E**nergy

Contents

1 Packages needed

- bluez5

2 Configuration

Init file modification for automatic Bluetooth start

/etc/udev/rules.d/10-local.rules:

```
# Set bluetooth power up
ACTION=="add", KERNEL=="hci0", RUN+="/usr/bin/hciconfig hci0 up"
```

3 Existing tool selection

Bluez provides some tools, by default, to analyze Bluetooth networks.
hciconfig to configure hci connections
hcitool to scan, find a device, connect to a device, manage a device list.. deviceS may be normal or low energy
gatttool for BLE device management

3.1 BLE device connection step-by-step

Command to scan all low-energy Bluetooth hardware:

```
Board $> hciconfig hci0 up
Board $> hcitool lescan
```

To scan available BLE devices:

```
Board $> hcitool lewladd <BLE_MAC_ADDRRESS>
```

To add BLE device in the white list (optional):

```
Board $> hcitool lecc <BLE_MAC_ADDRESS>
```

To connect a BLE device:

Once a BLE device is identified, its characteristics (attibutes) can be discovered, read and modified using GATTTOOL.
GATTTOOL offers two working modes: interractive and non-interractive

3.1.1 Gatttool Interractive mode

```
Board $> gatttool -b <MAC Address> --interactive
```

In interractive mode, a new prompt is available to perform BLE commands.
connect: to connect to a specified device
primary: to disable all primary attributes
char-read-hnd <handle> to read specified handle/attribute values
char-write-cmd <handle> <value> to modify handle values

3.1.2 Gatttool Non-interractive mode:

In non-interractive mode, commands are issued one by one. At each command, GATTTOOL performs device connection, action and disconnection.
Few a few examples are given below:

```
Board $> gatttool -b <Mac Address> --primary
Board $> gatttool -b <MAC Address> --characteristics
Board $> gatttool -b <MAC Address> --char-read
Board $> gatttool -b <MAC Address> --char-desc
```

Bluetooth Low Energy. Bluetooth LE, marketed as Bluetooth Smart is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group aimed at novel applications in the healthcare, fitness, beacons, security, and home entertainment industries.
Compared to Classic Bluetooth, Bluetooth Smart is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range. (sourcehttps://en.wikipedia.org/wiki/Bluetooth_Low_Energy)

Stable: 03.02.2020 - 08:42 / Revision: 03.02.2020 - 08:29

A quality version of this page, approved on *3 February 2020*, was based off this revision.

1 Scan for available Bluetooth devices

```
Board $> hcitool scan
Scanning ...
        A0:AF:BD:3B:26:61       lmecxl0923
        B0:55:08:40:33:84       HUAWEI P8 lite 2017
```

2 Setting the device to Visible

To control the visiblity of our Bluetooth device to other Bluetooth hardware.

- to enable visibility:

```
Board $> hciconfig hciX piscan
```

(hciX corresponds to the available Bluetooth hardware: hci0)

- To disable visibility:

```
Board $> hciconfig hciX noscan
```

(hciX corresponds to the available Bluetooth hardware: hci0)

3 How to scan via bluetoothd/systemd

Systemd provides a tool for Bluetooth management: bluetoothctl.
Example session with bluetoothclt for scanning, pairing, connecting:

```
Board $> bluetoothctl
[NEW] Controller 43:43:A1:12:1F:AC stm32mp1 [default]
Agent registered
[bluetooth]# power on
Changing power on succeeded
[CHG] Controller 43:43:A1:12:1F:AC Powered: yes
[bluetooth]# agent on
```

```
Agent is already registered
[bluetooth]# default-agent
Default agent request successful
[bluetooth]# scan on
Discovery started
[CHG] Controller 43:43:A1:12:1F:AC Discovering: yes
[NEW] Device D8:DB:36:D1:39:88 STM
...
[bluetooth]# scan off
[CHG] Controller 43:43:A1:12:1F:AC Discovering: no
Discovery stopped
[bluetooth]#pair D8:DB:36:D1:39:88
Pairing successful
[bluetooth]# connect D8:DB:36:D1:39:88
Connection successful
[STM]# quit
Agent unregistered
[DEL] Controller 43:43:A1:12:1F:AC stm32mp1 [default]
```

GPIO alternate function

System Trace Module

Stable: 03.02.2020 - 08:42 / Revision: 03.02.2020 - 08:29

A quality version of this page, approved on *3 February 2020*, was based off this revision.

1 How to setup Bluetooth

1.1 Setup

- Check that the driver has been probed correctly in the kernel log message

```
Board $>[    0.923697] STM32 USART driver initialized
Board $>[    0.928711] 4000e000.serial: ttyS1 at MMIO 0x4000e000 (irq = 42, base_baud =
6046875) is a stm32-usart
```

- Check the Bluetooth interface

```
Board $> hciconfig -a
hci0:   Type: Primary  Bus: UART
        BD Address: 43:43:A1:12:1F:AC  ACL MTU: 1021:8  SCO MTU: 64:1
        DOWN
        RX bytes:619 acl:0 sco:0 events:31 errors:0
        TX bytes:410 acl:0 sco:0 commands:31 errors:0
        Features: 0xbf 0xfe 0xcf 0xfe 0xdb 0xff 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH SNIFF
        Link mode: SLAVE ACCEPT
```

1.2 Initialize the Bluetooth interface

```
Board $> hciconfig hci0 up
Board $> hciconfig -a
hci0:   Type: Primary  Bus: UART
        BD Address: AA:AA:AA:AA:AA:AA  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING
        RX bytes:1378 acl:0 sco:0 events:74 errors:0
        TX bytes:1186 acl:0 sco:0 commands:74 errors:0
        Features: 0xbf 0xfe 0xcf 0xfe 0xdb 0xff 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH SNIFF
        Link mode: SLAVE ACCEPT
        Name: 'BlueZ 5.46'
        Class: 0x000000
        Service Classes: Unspecified
        Device Class: Miscellaneous,
        HCI Version: 4.1 (0x7)  Revision: 0x0
        LMP Version: 4.1 (0x7)  Subversion: 0x2209
        Manufacturer: Broadcom Corporation (15)
```

Universal Synchronous/Asynchronous Receiver/Transmitter

Universal Asynchronous Receiver/Transmitter

Automatic current limit (LCD power improvement solution)

Receive

Transmit

uniprocessor