



## USBH device tree configuration



## Contents

1 Article purpose .....	3
2 DT bindings documentation .....	4
3 DT configuration .....	5
3.1 DT configuration (STM32 level) .....	5
3.2 DT configuration (board level) .....	5
3.2.1 DT configuration with external high speed-HUB .....	6
3.2.2 DT configuration with OHCI to achieve full-speed and low-speed .....	6
3.3 DT configuration examples .....	6
3.3.1 DT configuration when using port0 with a high-speed hub .....	6
3.3.2 DT configuration when using port0 .....	6
3.3.3 DT configuration when using the two physical ports .....	7
4 How to configure the DT using STM32CubeMX .....	8
5 References .....	9



---

## 1 Article purpose

---

This article explains how to configure the **USBH** internal peripheral when it is assigned to the Linux<sup>®</sup>OS. In that case, it is controlled by the USB framework.

The configuration is performed using the *device tree* mechanism.

It is used by *USBH Linux drivers* (EHCI<sup>[1]</sup>, OHCI<sup>[2]</sup>) which register the relevant information in the USB framework.



---

## 2 DT bindings documentation

---

STM32 USBH internal peripheral is a USB Host device, composed of an **EHCI** controller and an **OHCI** controller.

Each controller is represented as a separate binding document:

- The generic USB EHCI controller device tree bindings<sup>[3]</sup> document deals with standard EHCI controller core resources (e.g. registers, clock, reset, interrupt, ...).
- The generic USB OHCI controller device tree bindings<sup>[4]</sup> document deals with standard OHCI controller core resources (e.g. registers, clock, reset, interrupt, ...).

Each controller uses the generic USB Host Controller Device (HCD) properties and generic USB properties, proposed by USB framework:

- The generic USB HCD device tree bindings<sup>[5]</sup> document deals with USB Host Controller PHY resources.
- The generic USB device tree bindings<sup>[6]</sup> document deals with USB optional properties (e.g. maximum speed, companion, ...).



### 3 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.

**STM32CubeMX** can be used to generate the board device tree. Refer to [How to configure the DT using STM32CubeMX](#) for more details.

#### 3.1 DT configuration (STM32 level)

The `usbh_ehci` and `usbh_ohci` DT nodes are declared in `stm32mp151.dtsi`<sup>[7]</sup>.

They are composed of a set of properties, used to describe the USBH\_EHCI and USBH\_OHCI controllers: registers address, clocks, resets, interrupts...

```
usbh_ohci: usbh-ohci@5800c000 {                                /* USBH OHCI controller */
    compatible = "generic-ohci";
    reg = <0x5800c000 0x1000>;
    clocks = <&rcc USBH>;
    resets = <&rcc USBH_R>;
    interrupts = <GIC_SPI 74 IRQ_TYPE_LEVEL_HIGH>;
    status = "disabled";
};

usbh_ehci: usbh-ehci@5800d000 {                                /* USBH EHCI controller */
    compatible = "generic-ehci";
    reg = <0x5800d000 0x1000>;
    clocks = <&rcc USBH>;
    resets = <&rcc USBH_R>;
    interrupts = <GIC_SPI 75 IRQ_TYPE_LEVEL_HIGH>;
    companion = <&usbh_ohci>;                                /* When USBH EHCI controller
detects a full or low speed device on a port,                * that port is switched
over to the USBH OHCI companion[6] controller */
    status = "disabled";
};
```



**This device tree part is related to STM32 microprocessors. It must be kept as is, without being modified by the end-user.**

#### 3.2 DT configuration (board level)

Follow the sequences described in the below chapters to configure and enable the USBH on your board.

USBH supports up to two physical ports, using on-chip USBPHYC:

- USBPHYC port#1 is assigned to the USBH
- USBPHYC port#2 can be statically assigned to the USBH or to the OTG



Please refer to [USBPHYC device tree configuration](#) for additional information on the USBPHYC configuration



USBH is composed of EHCI and OHCI controllers:

- high-speed operation is achieved through the EHCI controller
- full-speed and low-speed operation can be achieved by either:
  - a high-speed HUB IC wired to the downstream port
  - the OHCI controller

### 3.2.1 DT configuration with external high speed-HUB

- Enable the `usbh_ehci` by setting `status = "okay"`
- Configure the PHY(s) through `phys` and `phy-names`



There's no need to enable `usbh_ohci`, when all the low-speed and full-speed traffic is managed by the high-speed hub directly connected to the downstream port.

### 3.2.2 DT configuration with OHCI to achieve full-speed and low-speed

- Enable the `usbh_ehci` by setting `status = "okay"`
- Enable the `usbh_ohci` by setting `status = "okay"`

For both controllers:

- Configure the PHY(s) through `phys` and `phy-names`

## 3.3 DT configuration examples

### 3.3.1 DT configuration when using port0 with a high-speed hub

Below example shows how to configure the USBH when using the physical port 0 (the second physical port is unused).

A high-speed hub controller IC is used on the board: no need to enable `usbh_ohci`, all low-speed and full-speed traffic is managed by the hub.

```
&usbh_ehci {
    phys = <&usbphyc_port0>;          /* Use USBPHYC HS PHY port #1, mapped on USBH
    controller */
    phy-names = "usb";
    status = "okay";
};

/* No need to configure and enable usbh_ohci */
```

### 3.3.2 DT configuration when using port0

Below example shows how to configure the USBH when using the physical port 0 (the second physical port is unused, or in use by the OTG and no high-speed hub is used).

```
&usbh_ehci {
    phys = <&usbphyc_port0>;          /* Use USBPHYC HS PHY port #1, mapped on USBH
    controller */
    phy-names = "usb";
    status = "okay";
};
```



```
&usbh_ohci {
    phys = <&usbphyc_port0>;          /* Use USBPHYC HS PHY port #1, mapped on USBH
    controller */
    phy-names = "usb";
    status = "okay";
};
```

### 3.3.3 DT configuration when using the two physical ports

Below example shows how to configure the USBH when using the two physical ports.

```
&usbh_ehci {
    phys = <&usbphyc_port0>, <&usbphyc_port1 1>; /* Use USBPHYC HS PHY port #1,
    mapped on USBH controller */                /* Use USBPHYC HS PHY port #2, co
    nfigure UTMI switch to select USBH controller */
    phy-names = "usb", "usb";
    status = "okay";
};

&usbh_ohci {
    phys = <&usbphyc_port0>, <&usbphyc_port1 1>; /* Use USBPHYC HS PHY port #1,
    mapped on USBH controller */                /* Use USBPHYC HS PHY port #2, co
    nfigure UTMI switch to select USBH controller */
    phy-names = "usb", "usb";
    status = "okay";
};
```



---

## 4 How to configure the DT using STM32CubeMX

---

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

---

Please refer to the following links for full description:

- [drivers/usb/host/ehci-platform.c](#) , Generic platform ehci driver
- [drivers/usb/host/ohci-platform.c](#) , Generic platform ohci driver
- [Documentation/devicetree/bindings/usb/generic-ehci.yaml](#) Generic USB EHCI controller device tree bindings
- [Documentation/devicetree/bindings/usb/generic-ohci.yaml](#) Generic USB OHCI controller device tree bindings
- [Documentation/devicetree/bindings/usb/usb-hcd.yaml](#) Generic USB HCD (Host Controller Device) device tree bindings
- [6.06.1 Documentation/devicetree/bindings/usb/generic.txt](#) Generic USB device tree bindings
- [arch/arm/boot/dts/stm32mp151.dtsi](#) , STM32MP151 device tree file

Linux<sup>®</sup> is a registered trademark of Linus Torvalds.

Operating System

USB Host (STM32 specific)

Enhanced Host Controller Interface

Open Host Controller Interface

Device Tree

Generic Interrupt Controller

Serial Peripheral Interface

High Speed (MIPI<sup>®</sup> Alliance DSI standard)

USB 2.0 Transceiver Macrocell Interface