OTG device tree configuration

Contents

1. OTG device tree configuration 3
2. Device tree 11
3. OTG internal peripheral 11
4. Pinctrl device tree configuration 11
5. Pinctrl overview 11
6. Regulator overview 11
7. STM32CubeMX 11
8. USB overview 12
9. USBH internal peripheral 12
10. USBPHYC device tree configuration 12
11. USBPHYC internal peripheral 12

Stable: 26.03.2021 - 15:53 / Revision: 19.03.2021 - 13:15

Stable: 20.06.2021 - 10.0.0 / Revision: 10.06.2021 - 10.1.0

Contents

1 Article purpose 4
2 DT bindings documentation 5
3 DT configuration 6
3.1 DT configuration (STM32 level) 6
3.2 DT configuration (board level) 6
3.2.1 DT configuration using full-speed USB PHY 6
3.2.2 DT configuration using high-speed USB PHY 7
3.3 DT configuration examples 7
3.3.1 DT configuration example as full-speed OTG, with micro-B connector 7
3.3.2 DT configuration example as high speed OTG, with micro-B connector 8
3.3.3 DT configuration example as high speed OTG, with Type-C connector 8
4 How to configure the DT using STM32CubeMX 10
5 References 11

1 Article purpose

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

The configuration is performed using the device tree mechanism.

It is used by *OTG Linux driver*[1] which registers the relevant information in the USB framework.

2 DT bindings documentation

The *Platform DesignWare HS OTG USB 2.0 controller device tree bindings*[2] document represents the OTG (DRD) controller.

The *Generic USB device tree bindings*[3] document represents generic USB properties, proposed by USB framework such as maximum speed, dr_mode...

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 **usbotg_hs** DT node is declared in stm32mp151.dtsi[4].

It is composed of a set of properties, used to describe the OTG controller: registers address, clocks, resets, interrupts...

```
usbotg_hs: usb-otg@49000000 {
        compatible = "snps,dwc2";
        reg = <0x49000000 0x10000>;
        clocks = <&rcc USBO_K>;
        clock-names = "otg";
        resets = <&rcc USBO_R>;
        reset-names = "dwc2";
        interrupts = <GIC_SPI 98 IRQ_TYPE_LEVEL_HIGH>;
        dr_mode = "otg";                                    /* dr_mode[3] can be
overwritten at board level to set a particular mode */
        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 OTG on your board.

OTG supports two PHY interfaces that can be statically selected via DT:

- full-speed PHY, embedded with the OTG controller
- high-speed USBPHYC HS PHY that can be assigned to either the USBH or the OTG controller.

When operating in "otg" or "host" mode, an external charge pump, e.g. 5V regulator must be specified.

Please refer to Regulator overview for additional information on regulators configuration.

3.2.1 DT configuration using full-speed USB PHY

- Enable the OTG by setting **status = "okay".**
- Use embedded full-speed PHY by setting **compatible = "st,stm32mp1-fsotg", "snps,dwc2"**
- Configure full-speed PHY pins (OTG ID, DM, DP) as analog via pinctrl, through **pinctrl-0** and **pinctrl-names**.

- Optionally set dual-role mode through **dr_mode = "peripheral"**, **"host"** or **"otg"** (default to "otg" in case it is not set)
- Optionally set vbus voltage regulator for otg and host modes, through **vbus-supply = <&your_regulator>**

3.2.2 DT configuration using high-speed USB PHY

- Enable the OTG by setting **status = "okay".**
- Select USBPHYC port#2 by setting **phys = <&usbphyc_port1 0>;** and **phy-names = "usb2-phy";**
- Optionally configure OTG ID pin as analog via pinctrl, through **pinctrl-0** and **pinctrl-names**.
- Optionally set dual-role mode through **dr_mode = "peripheral"**, **"host"** or **"otg"** (default to "otg" in case it is not set)
- Optionally set vbus voltage regulator for otg and host modes, through **vbus-supply = <&your_regulator>**

Please refer to USBPHYC device tree configuration for additional information on USBPHYC port#2 configuration

3.3 DT configuration examples

3.3.1 DT configuration example as full-speed OTG, with micro-B connector

The example below shows how to configure full-speed OTG, with the ID pin to detect role (peripheral, host):

- OTG ID and data (DM, DP) pins: use Pinctrl device tree configuration to configure PA10, PA11 and PA12 as analog input.
- Use integrated *full-speed USB PHY* by setting **compatible**
- Dual-role (dr_mode) is "otg" (e.g. the default as unspecified)
- Use vbus voltage regulator

```
# part of pin-controller dt node
usbotg_hs_pins_a: usbotg_hs-0 {
        pins {
                pinmux = <STM32_PINMUX('A', 10, ANALOG)>; /* OTG_ID */            /*
configure 'PA10' as ANALOG */

        };
};

usbotg_fs_dp_dm_pins_a: usbotg-fs-dp-dm-0 {
        pins {
                pinmux = <STM32_PINMUX('A', 11, ANALOG)>, /* OTG_FS_DM */
                         <STM32_PINMUX('A', 12, ANALOG)>; /* OTG_FS_DP */
        };
};
```

```
&usbotg_hs {
        compatible = "st,stm32mp1-fsotg", "snps,dwc2";                          /* Use
full-speed integrated PHY */
        pinctrl-names = "default";
        pinctrl-0 = <&usbotg_hs_pins_a &usbotg_fs_dp_dm_pins_a>;                /*
configure OTG ID and full-speed data pins */
        vbus-supply = <&vbus_otg>;                                              /*
voltage regulator to supply Vbus */
        status = "okay";
};
```

3.3.2 DT configuration example as high speed OTG, with micro-B connector

The example below shows how to configure high-speed OTG, with the ID pin to detect role (peripheral, host):

- OTG ID pin: use Pinctrl device tree configuration to configure PA10 as analog input.
- Use *USB HS PHY port#2*, with the UTMI switch that selects the OTG controller
- Dual-role mode (dr_mode) is "otg" (e.g. the default as unspecified)
- Use vbus voltage regulator

```
# part of pin-controller dt node
usbotg_hs_pins_a: usbotg_hs-0 {
        pins {
                pinmux = <STM32_PINMUX('A', 10, ANALOG)>; /* OTG_ID */            /*
configure 'PA10' as ANALOG */

        };
};
```

```
&usbotg_hs {
        compatible = "st,stm32mp1-hsotg", "snps,dwc2";
        pinctrl-names = "default";
        pinctrl-0 = <&usbotg_hs_pins_a>;                                          /*
configure OTG_ID pin */
        phys = <&usbphyc_port1 0>;                                                /* 0: UT
MI switch selects the OTG controller */
        phy-names = "usb2-phy";
        vbus-supply = <&vbus_otg>;                                                /*
voltage regulator to supply Vbus */
        status = "okay";                                                          /*
enable OTG */
};
```

3.3.3 DT configuration example as high speed OTG, with Type-C connector

The example below shows how to configure high-speed OTG with a Type-C connector. Type-C is managed by an external controller which detects connection and data role (peripheral, host) and implements Linux USB role switch class:

- Use *USB HS PHY port#2*, with the UTMI switch that selects the OTG controller
- Dual-role mode (dr_mode) is "otg" (e.g. the default as unspecified)
- Add usb-role-switch property to OTG controller node: it indicates that the device is capable of assigning the USB data role (USB host or USB device) for a given USB connector.
- Add a connector subnode to the Type-C controller node, with a port child node pointing to the OTG controller endpoint and add a port child node to the OTG controller node, pointing to the Type-C controller endpoint: Type-C controller driver will be able to get the USB role switch to inform it of a role change.

```
#example of Type-C controller node
stusb1600@28 {
        compatible = "st,stusb1600";
        reg = <0x28>;
        interrupts = <11 IRQ_TYPE_EDGE_FALLING>;
        interrupt-parent = <&gpioi>;
        pinctrl-names = "default";
        pinctrl-0 = <&stusb1600_pins_a>;
        status = "okay";
        vdd-supply = <&vin>;
```

```
        connector {
                compatible = "usb-c-connector";
                label = "USB-C";
                power-role = "dual";
                power-opmode = "default";
```

```
                port {
                        con_usbotg_hs_ep: endpoint {
                                remote-endpoint = <&usbotg_hs_ep>;                    /*
point the OTG controller endpoint node */
                        };
                };
        };
};
```

```
&usbotg_hs {
        compatible = "st,stm32mp1-hsotg", "snps,dwc2";
        phys = <&usbphyc_port1 0>;                                                    /* 0: UT
MI switch selects the OTG controller */
        phy-names = "usb2-phy";
        usb-role-switch;                                                              /* see
USB generic bindings [5] */
        status = "okay";                                                              /*
enable OTG */

        port {
                usbotg_hs_ep: endpoint {
                        remote-endpoint = <&con_usbotg_hs_ep>;                      /*
point the Type-C controller endpoint node */
                };
        };
};
```

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 additional information:

- drivers/usb/dwc2/ , DesignWare HS OTG Controller driver
- Documentation/devicetree/bindings/usb/dwc2.yaml , Platform DesignWare HS OTG USB 2.0 controller device tree bindings
- 3.03.1 Documentation/devicetree/bindings/usb/generic.txt , Generic USB device tree bindings
- arch/arm/boot/dts/stm32mp151.dtsi , STM32MP151 device tree file
- Documentation/devicetree/bindings/usb/generic.txt , USB generic bindings

Operating System

USB On-The-Go (Capability/type of USB port, acting primarily as USB device, to also act as USB host. Also known as USB OTG.)

Device Tree

High Speed (MIPI® Alliance DSI standard)

Dual-Role Device (USB standard defines host and device roles. OTG controllers support both roles and can be called Dual-Role Devices controllers.)

Generic Interrupt Controller

Serial Peripheral Interface

USB 2.0 Transceiver Macrocell Interface

Stable: 19.03.2021 - 08:52 / Revision: 19.03.2021 - 08:49

Invalid target: no reviewed revision corresponds to the given ID.

Return to Device tree.

Stable: 26.03.2021 - 15:54 / Revision: 18.03.2021 - 14:35

Invalid target: no reviewed revision corresponds to the given ID.

Return to OTG internal peripheral.

Stable: 11.06.2020 - 09:00 / Revision: 10.06.2020 - 15:35

Invalid target: no reviewed revision corresponds to the given ID.

Return to Pinctrl device tree configuration.

Stable: 11.06.2020 - 09:03 / Revision: 10.06.2020 - 15:17

Invalid target: no reviewed revision corresponds to the given ID.

Return to Pinctrl overview.

Stable: 11.06.2020 - 12:50 / Revision: 11.06.2020 - 12:12

Invalid target: no reviewed revision corresponds to the given ID.

Return to Regulator overview.

Stable: 23.09.2020 - 13:22 / Revision: 12.06.2020 - 13:25

Invalid target: no reviewed revision corresponds to the given ID.

Return to STM32CubeMX.
Stable: 26.03.2021 - 15:22 / Revision: 18.03.2021 - 14:33

Invalid target: no reviewed revision corresponds to the given ID.

Return to USB overview.
Stable: 25.09.2020 - 09:43 / Revision: 25.09.2020 - 09:37

Invalid target: no reviewed revision corresponds to the given ID.

Return to USBH internal peripheral.
Stable: 26.03.2021 - 15:16 / Revision: 19.03.2021 - 13:35

Invalid target: no reviewed revision corresponds to the given ID.

Return to USBPHYC device tree configuration.
Stable: 25.09.2020 - 09:43 / Revision: 25.09.2020 - 09:39

Invalid target: no reviewed revision corresponds to the given ID.

Return to USBPHYC internal peripheral.