Serial TTY device tree configuration

Serial TTY device tree configuration

Stable: 10.04.2020 - 15:13 / Revision: 10.04.2020 - 15:12

Contents

1 Article Purpose

This article explains how to configure the USART when it is assigned to the Linux®OS. In that case, it is controlled by the Serial and TTY frameworks.

The configuration is performed using the device tree mechanism that provides a hardware description of the USART peripheral, used by the stm32-usart Linux driver.

If the peripheral is assigned to another execution context, refer to How to assign an internal peripheral to a runtime context article for guidelines on peripheral assignment and configuration.

2 DT bindings documentation

The USART is a multifunction device.

Each function is represented by a separate binding document:

- Generic UART bindings[1] used by UART framework.
- STM32 USART driver bindings[2] used by stm32-usart driver. This bindings documentation explains how to write device tree files for STM32 USARTs.

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)

All STM32 USART nodes (excepted USART1, secure instance under ETZPC control) are described in microprocessor device tree (ex: stm32mp157c.dtsi [3]) with default parameters and disabled status.
The required and optional properties are fully described in the bindings files.

This device tree configuration related to the STM32 should be kept as is, without being modified by the customer.

3.2 DT configuration (Board level)

Part of the device tree is used to describe the USART hardware used on a given board:

- Which USART instances are enabled (by setting status to "okay")
- Which features are used (such as DMA transfer or direct transfer, transfer speed or parity)
- Which pins are configured via pinctrl.
- Which serial aliases are linked to UART instances. Please check the alias already used in other device tree files to avoid alias conflicts. The alias defines the index of the ttySTMx instance linked the UART.

Note:

- As the pin configuration can be different for each board, several DT configurations can be defined for each UART instance.
- The pin configuration is described in board datasheet. Each new pin configuration described in boards datasheet needs to be defined in device tree.

Four device tree configurations can be defined for each pin muxing configuration:

- Default: for standard usage (mandatory)
- "sleep": for Sleep mode, when the UART instance is not a wake up source (mandatory)
- "idle": for Sleep mode, when the UART instance is a wake up source (optional)
- "no_console_suspend": when the UART instance is used for the console, and no_console_suspend mode is activated (optional).

3.3 DT configuration examples

3.3.1 Activation of a USART or UART instance

Some UART pins are available on GPIO expansion and Arduino connectors (depending on the connectors available on the board).

- STM32MP157C-EV1 Evaluation board GPIO expansion connector
- STM32MP157X-DKX Discovery kit GPIO expansion connector

To communicate with a UART instance, an RS232 card must be plugged on the UART pins.

The example below shows how to configure and enable a UART instance at board level, based on STM32MP157C-EV1 board USART3 example.
Note: For STM32 boards, the configuration is already defined in the device tree. Only the device activation is needed.

To activate a UART instance, please follow steps below:

- Define the instance pin configuration (ex: stm32mp157-pinctrl.dtsi [4]).

```
usart3_pins_a: usart3-0 {
        pins1 {
                /* USART3 TX and RTS pins activation for default mode */
                pinmux = <STM32_PINMUX('B', 10, AF7)>, /* USART3_TX */
                         <STM32_PINMUX('G', 8, AF8)>; /* USART3_RTS */
                bias-disable;
                drive-push-pull;
                slew-rate = <0>;
        };
        pins2 {
                /* USART3 RX and CTS_NSS pins activation for default mode */
                pinmux = <STM32_PINMUX('B', 12, AF8)>, /* USART3_RX */
                         <STM32_PINMUX('I', 10, AF8)>; /* USART3_CTS_NSS */
                bias-disable;
        };
};

usart3_idle_pins_a: usart3-idle-0 {
        pins1 {
                /* USART3 TX, RTS, and CTS_NSS pins deactivation for sleep mode */
                pinmux = <STM32_PINMUX('B', 10, ANALOG)>, /* USART3_TX */
                         <STM32_PINMUX('G', 8, ANALOG)>, /* USART3_RTS */
                         <STM32_PINMUX('I', 10, ANALOG)>; /* USART3_CTS_NSS */
        };
        pins2 {
                /* USART3_RX pin still active for wake up */
                pinmux = <STM32_PINMUX('B', 12, AF8)>; /* USART3_RX */
                bias-disable;
        };
};

usart3_sleep_pins_a: usart3-sleep-0 {
        pins {
                /* USART3_TX, RTS, CTS_NSS, and RX pins deactivation for sleep mode */
                pinmux = <STM32_PINMUX('B', 10, ANALOG)>, /* USART3_TX */
                         <STM32_PINMUX('G', 8, ANALOG)>, /* USART3_RTS */
                         <STM32_PINMUX('I', 10, ANALOG)>, /* USART3_CTS_NSS */
                         <STM32_PINMUX('B', 12, ANALOG)>; /* USART3_RX */
        };
};
```

- Define the serial alias for this instance at board level (ex: stm32mp157c-ev1.dts [5]).

```
aliases {
        /* Serial1 alias (ie ttySTM1) assigned to usart3 */
        serial1 = &usart3;
        ethernet0 = &ethernet0;
};
```

- Configure and activate the instance at board level (ex: stm32mp157c-ev1.dts [5]).

```
&usart3 {
        pinctrl-names = "default", "sleep", "idle";        /* pin configurations
definition */
        pinctrl-0 = <&usart3_pins_a>;                        /* default pin
configuration selection */
        pinctrl-1 = <&usart3_sleep_pins_a>;                /* sleep pin configuration
selection */
        pinctrl-2 = <&usart3_idle_pins_a>;                /* idle pin configuration
selection */
        status = "okay";                                /* device activation */
};
```

Note: The pin configuration selected has to be aligned with the pin configuration described in the board datasheet.

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

- Documentation/devicetree/bindings/serial/serial.txt , UART generic device tree bindings
- Documentation/devicetree/bindings/serial/st,stm32-usart.txt , STM32 USART device tree bindings
- arch/arm/boot/dts/stm32mp157c.dtsi , STM32MP157C device tree file
- arch/arm/boot/dts/stm32mp157-pinctrl.dtsi , STM32MP157 pinctrl device tree file
- 5.05.1 arch/arm/boot/dts/stm32mp157c-ev1.dts , STM32MP157c ev1 board device tree file

Operating System

Universal Synchronous/Asynchronous Receiver/Transmitter

Device Tree

Universal Asynchronous Receiver/Transmitter

Direct Memory Access

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.)

Transmit

Receive

Compatibility Test Suite (Android specific) or Clear to send (in UART context)