DAC device tree configuration

Contents

Stable: 04.11.2020 - 13:17 / Revision: 04.11.2020 - 13:14

Stable: 04.11.2020 - 10:17 / Revision: 04.11.2020 - 10:17

A quality version of this page, approved on *4 November 2020*, was based off this revision.

Contents

1 Purpose

The purpose of this article is to explain how to configure the digital-to-analog converter (*DAC*)[1] **when the peripheral is assigned to Linux®OS**, and in particular:

- how to configure and enable the DAC **peripheral**
- how to configure the **board**, the DAC channels, reference voltage regulator and pins.

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

It is used by the DAC Linux driver that registers relevant information in IIO framework, such as IIO devices, channels and voltage scale.

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

STM32 DAC device tree bindings[3] deal with all the required or optional properties.

3 DT configuration

This hardware description is a combination of STM32 and board device tree files. See Device tree for explanations on device tree file split.

The **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 DAC nodes are declared in stm32mp151.dtsi[4]:

- **DT root node** ('**dac**') describes the DAC hardware block parameters such as registers area and clocks.
- **DT child nodes** ('**dac1**' and '**dac2**') describe DAC channels independently.

```
dac: dac@address {
        compatible = "st,stm32h7-dac-core";
        ...                                                                     /*
common resources in 'dac' root node. */
        dac1: dac@1 {
                compatible = "st,stm32-dac";
                reg = <1>;                                                      /*
DAC identifier (e.g. 1 for DAC1) */
                ...                                                             /*
private resources in 'dac1' child node. */
        };
        dac2: dac@2 {
                compatible = "st,stm32-dac";
                reg = <2>;                                                      /*
DAC identifier (e.g. 2 for DAC2) */
                ...                                                             /*
private resources in 'dac2' child node. */
        };
};
```

Warning

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

3.2 DT configuration (board level)

Follow the below sequence to configure and enable the DAC on your board:

- Enable **DT root node** named '**dac**' by setting **status = "okay"**.
- Configure pins in use via pinctrl through **pinctrl-0** and **pinctrl-names**.
- Configure analog reference voltage regulator[5] by setting **vref-supply = <&your_regulator>**.
- Enable **DT child node(s)** for '**dac1**' and/or '**dac2**' channels(s) in use by setting **status = "okay"**.

Information

The DAC can use the internal VREFBUF[6] or any other external regulator[5] wired to the VREF+ pin

.

3.3 DT configuration example

The example below shows how to configure DAC1 and DAC2 channels:

- PA4 and PA5 pins both configured as analog pins (see Pinctrl device tree configuration for more details)
- VREFBUF[6] used as reference voltage

```
dac_ch1_pins_a: dac-ch1 {
        pins {
                pinmux = <STM32_PINMUX('A', 4, ANALOG)>;                /* configure
'PA4' as ANALOG */
        };
};
dac_ch2_pins_a: dac-ch2 {
        pins {
                pinmux = <STM32_PINMUX('A', 5, ANALOG)>;                /* configure
'PA5' as ANALOG */
        };
};
```

```
&dac {
        pinctrl-names = "default";
        pinctrl-0 = <&dac_ch1_pins_a &dac_ch2_pins_a>;                  /* Use PA4 and
PA5 pin as ANALOG */
        vref-supply = <&vrefbuf>;                                       /* Example to
use VREFBUF (It needs to be enabled as well) */
        status = "okay";                                                /* Enable the DAC
 block */
        dac1: dac@1 {
                status = "okay";                                        /* Enable DAC1 */
        };
        dac2: dac@2 {
                status = "okay";                                        /* Enable DAC2 */
        };
};
```

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

For additional information, refer to the following links:

- DAC internal peripheral
- Device tree
- Documentation/devicetree/bindings/iio/dac/st,stm32-dac.txt , STM32 DAC device tree bindings
- STM32MP151 device tree file
- 5.05.1 Regulator overview
- 6.06.1 VREFBUF internal peripheral

Digital-to-analog converter (Electronic circuit that converts a binary number into a continuously varying value.)

Linux® is a registered trademark of Linus Torvalds.

Operating System

Industrial I/O Linux subsystem

Device Tree

voltage reference buffer (STM32 specific)