BSEC device tree configuration

Stable: 09.10.2019 - 10:20 | Revision: 09.10.2019 - 10:00

A quality version of this page, accepted on *9 October 2019*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure BSEC at boot time.

This article describes the BSEC configuration, which is performed using the device tree mechanism that provides a hardware description of the BSEC peripheral.

2 DT bindings documentation

Generic information about NVMEM is available in NVMEM overview.

This binding document explains how to write device tree files for BSEC:

- TF-A: *tf-a/docs/devicetree/bindings/soc/st,stm32-romem.txt"*[1]
- Linux®BSEC devicetree bindings: Documentation/devicetree/bindings/nvmem/st,stm32-romem.txt[2]
- Linux® generic NVMEM devicetree bindings: Documentation/devicetree/bindings/nvmem/nvmem.txt[3]

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 STM32MP1 BSEC node is located in *stm32mp157c.dtsi*[4] (see Device tree for more explanations).

```
/ {
...
        soc {
...
                bsec: nvmem@5c005000 {
                        compatible = "st,stm32mp15-bsec";
                        reg = <0x5c005000 0x400>;
                        #address-cells = <1>;
                        #size-cells = <1>;
                        ts_cal1: calib@5c {
                                reg = <0x5c 0x2>;
                        };
                        ts_cal2: calib@5e {
                                reg = <0x5e 0x2>;
                        };
                };
...
        };
...
};
```

Please refer to NVMEM overview for the bindings common with Linux® kernel.

3.2 DT configuration (board level)

3.2.1 STM32MP1 BSEC node append

Board definition in Device tree may add some OTP declarations, specific to the board:

For ecosystem release v1.1.0

```
&bsec {
        board_id: board_id@ec {
                reg = <0xec 0x4>;
                status = "okay";
        };
};
```

Upper OTPs are supposed to contain sensitive data such as keys or passwords. But with only 32 lower NVMEM 32-bit data words, software may need more, so it is possible to manage exceptions in order to allow some upper OTPs to be accessed by non-secure world, through secure world services for very specific needs.
User can add upper OTP declaration in device tree, using status property, to define accessibility conditions, as described in the following table:

status	Upper OTP available from
disabled	secure only (normal behavior)
okay	non-secure and secure (exception)

When status property is not filled, this is implicitly set as an "okay" status by default.

secure-status property can appear in some OTP declarations, please don't care.

For ecosystem release v1.0.0

```
&bsec {
        board_id: board_id@ec {
                reg = <0xec 0x4>;
        };
};
```

As in previous section, exceptions are managed, but they are only checked in case of closed_device BSEC mode. In open_device mode, all upper OTPs non-secure accesses are allowed. See STM32MP15 reference manuals for more information about these modes.

3.2.2 STM32MP1 BSEC node append (bootloader specific)

The bootloader specific STM32MP1 BSEC node append data is located in *stm32mp157c-security.dtsi*[5] (see Device tree for more explanations).

```
&bsec {
        mac_addr: mac_addr@e4 {
                reg = <0xe4 0x6>;
        };
        /* Spare field to align on 32-bit OTP granularity  */
        spare_ns_ea: spare_ns_ea@ea {
                reg = <0xea 0x2>;
        };
};
```

3.2.3 STM32MP1 driver node append

Driver can directly consume NVMEM data cells, as described in NVMEM overview.
The ADC_TEMP device is a good example, with a dedicated OTP containing calibration information.
The device node is located in *stm32mp157c.dtsi*[6] file.

```
adc_temp: temp {
        compatible = "st,stm32mp1-adc-temp";
        io-channels = <&adc2 12>;
        nvmem-cells = <&ts_cal1>, <&ts_cal2>;
        nvmem-cell-names = "ts_cal1", "ts_cal2";
        #io-channel-cells = <0>;
        #thermal-sensor-cells = <0>;
        status = "disabled";
};
```

With these nvmem-cells / nvmem-cell-names properties, the ADC_TEMP device can easily find the OTP number, in order to access calibration information.

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:

- docs/devicetree/bindings/soc/st,stm32-romem.txt TF-A BSEC binding information file
- Documentation/devicetree/bindings/nvmem/st,stm32-romem.txt
- Documentation/devicetree/bindings/nvmem/nvmem.txt
- fdts/stm32mp157c.dtsi (for TF-A): STM32MP157C device tree files
- fdts/stm32mp157c-security.dtsi (for TF-A): STM32MP157C device tree files
- arch/arm/boot/dts/stm32mp157c.dtsi

Template:ArticleMainWriter

Device Tree

Boot and Security and OTP control

One Time Programmed

Analog-to-digital converter. The process of converting a sampled analog signal to a digital code that represents the amplitude of the original signal sample.