BSEC device tree configuration

Contents

Stable: 05.11.2020 - 10:19 / Revision: 04.11.2020 - 10:01

A quality version of this page, approved on *5 November 2020*, 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 performed using the device tree mechanism, which provides a hardware description of the BSEC peripheral.

2 DT bindings documentation

Generic information about NVMEM is available in the NVMEM overview.

The following binding-related documentation 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.yaml[3] and Documentation /devicetree/bindings/nvmem/nvmem-consumer.yaml[4]

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 the file *stm32mp151.dtsi*[5] (see Device tree for further explanation).

```
/ {
...
        soc {
...
                bsec: nvmem@5c005000 {
                        compatible = "st,stm32mp15-bsec";
                        reg = <0x5c005000 0x400>;
                        #address-cells = <1>;
                        #size-cells = <1>;

                        part_number_otp: part_number_otp@4 {
                                reg = <0x4 0x1>;
                        };
                        ts_cal1: calib@5c {
                                reg = <0x5c 0x2>;
                        };
                        ts_cal2: calib@5e {
                                reg = <0x5e 0x2>;
                        };
                };
...
        };
...
};
```

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

3.2 DT configuration (board level)

3.2.1 STM32MP1 BSEC node append

The board definition in the device tree may include some additional board-specific OTP declarations:

```
&bsec {
        board_id: board_id@ec {
                reg = <0xec 0x4>;
                st,non-secure-otp;
        };
};
```

With only 32 lower NVMEM 32-bit data words, the software needs to manage exceptions in order to allow some upper OTPs to be accessed by the non-secure world, through secure world services for very specific needs. The user can add an OTP declaration in the device tree, using the "st,non-secure-otp" property, with a 32-bit length granularity (that is, 4 bytes).

3.2.2 STM32MP1 BSEC node append (bootloader specific)

The bootloader-specific STM32MP1 BSEC node append data is located in the file *stm32mp151.dtsi* [6] for TF-A (see Device tree for further explanation).
This completes NVMEM data providers, for bootloader-specific purposes only, either for a driver, or the platform itself.

```
bsec: nvmem@5c005000 {
        compatible = "st,stm32mp15-bsec";
        reg = <0x5c005000 0x400>;
        #address-cells = <1>;
        #size-cells = <1>;

        cfg0_otp: cfg0_otp@0 {
                reg = <0x0 0x1>;
        };
        part_number_otp: part_number_otp@4 {
                reg = <0x4 0x1>;
        };
        monotonic_otp: monotonic_otp@10 {
                reg = <0x10 0x4>;
        };
        nand_otp: nand_otp@24 {
                reg = <0x24 0x4>;
        };
        uid_otp: uid_otp@34 {
                reg = <0x34 0xc>;
        };
        package_otp: package_otp@40 {
                reg = <0x40 0x4>;
        };
        hw2_otp: hw2_otp@48 {
                reg = <0x48 0x4>;
        };
        ts_cal1: calib@5c {
                reg = <0x5c 0x2>;
        };
        ts_cal2: calib@5e {
                reg = <0x5e 0x2>;
        };
        pkh_otp: pkh_otp@60 {
                reg = <0x60 0x20>;
        };
        mac_addr: mac_addr@e4 {
                reg = <0xe4 0x8>;
                st,non-secure-otp;
        };
};
```

Please see the "st,non-secure-otp" definition in the previous section above. No more spare field declaration here.

3.2.3 STM32MP1 driver node append

The driver can directly consume NVMEM data cells, as described in NVMEM overview.
The CPU0 device is a good example, with a dedicated OTP containing part number information.
The device node is located in the *stm32mp151.dtsi*[5] file.

```
cpu0: cpu@0 {
        compatible = "arm,cortex-a7";
        device_type = "cpu";
        reg = <0>;
        clocks = <&scmi0_clk CK_SCMI0_MPU>;
        clock-names = "cpu";
        operating-points-v2 = <&cpu0_opp_table>;
        nvmem-cells = <&part_number_otp>;
        nvmem-cell-names = "part_number";
        #cooling-cells = <2>;
};
```

With these nvmem-cells / nvmem-cell-names properties, the CPU0 device can easily find the OTP number, in order to access part number information.

3.2.4 STM32MP1 nvmem_layout node (bootloader specific)

The STM32MP1 nvmem_layout node gathers all NVMEM platform-dependent layout information, including OTP names and phandles, in order to allow easy access for data consumers, using pre-defined string in the nvmem-cell-names property.

```
nvmem_layout: nvmem_layout@0 {
        compatible = "st,stm32mp1-nvmem-layout";
        nvmem-cells = <&cfg0_otp>,
                      <&part_number_otp>,
                      <&monotonic_otp>,
                      <&nand_otp>,
                      <&uid_otp>,
                      <&package_otp>,
                      <&hw2_otp>;

        nvmem-cell-names = "cfg0_otp",
                           "part_number_otp",
                           "monotonic_otp",
                           "uid_otp",
                           "nand_otp",
                           "package_otp",
                           "hw2_otp";
};
```

With this new node, the platform can easily find the OTP numbers, in order to access all the necessary 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.
STM32CubeMX may not support all the properties described in the documents listed in DT bindings documentation above. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties that are preserved from one generation to another. Refer to the 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.yaml
- Documentation/devicetree/bindings/nvmem/nvmem-consumer.yaml
- 5.05.1 arch/arm/boot/dts/stm32mp151.dtsi : STM32MP151 Linux kernel device tree files
- fdts/stm32mp151.dtsi STM32MP151 TF-A device tree files

Device Tree

Boot and Security and OTP control

Linux® is a registered trademark of Linus Torvalds.

One Time Programmed

Microprocessor Unit

Trusted Firmware for Arm® Cortex®-A