Ethernet device tree configuration

Contents

Stable: 06.10.2020 - 13:06 / Revision: 06.10.2020 - 12:35

Stable: 06.10.2020 - 10:00 | Revision: 06.10.2020 - 12:38

A quality version of this page, approved on *6 October 2020*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the Ethernet when it is assigned to the Linux®OS. In that case, it is controlled by the Ethernet framework

The configuration is performed using the device tree mechanism that provides a hardware description of the Ethernet peripheral, used by the STM32 DWMAC driver

2 DT bindings documentation

The *Ethernet* is a multifunction device.

Each function is represented by a separate binding document:

- "Generic" Ethernet device tree bindings [1]
- specific STM32 ETH device tree bindings[2]

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.

3.1 DT configuration (STM32 level)

Ethernet peripheral nodes are located in stm32mp151.dtsi [3] file with a disabled status and some required properties such as:

- Physical base address and size of the device register map
- STMMAC interrupts
- stmmaceth clock and Rx, Tx clocks

This is a set of properties that may not vary for a given STM32MP device, such as: register addresses, interrupts, clocks, ...

```
ethernet0: ethernet@5800a000 {
        compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
        reg = <0x5800a000 0x2000>;
        reg-names = "stmmaceth";
        interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_NONE>;
        interrupt-names = "macirq";
        clock-names = "stmmaceth",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
        snps,pbl = <2>;
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

The required and optional properties are fully described in the bindings files.

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

3.2 Ethernet DT configuration (board level)

The device tree board file (.dts) contains all hardware configurations related to board design. The DT node (**"ethernet"**) should be updated to:

- Enable the Ethernet block by setting **status = "okay".**
- Configure the pins in use via pinctrl, through **pinctrl-0** (default pins), **pinctrl-1** (sleep pins) and **pinctrl-names**.
- Configure Ethernet interface used **phy-mode = "rgmii".**, (rmii, mii, gmii).
- Configure Ethernet max speed **max-speed = <1000>".**.

```
&ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rgmii_pins_a>;
        pinctrl-1 = <&ethernet0_rgmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rgmii";
        max-speed = <1000>;
        phy-handle = <&phy0>;

        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@1 {
                        reg = <1>;
                };
        };
};
```

3.3 DT configuration examples

The example below shows how to configure and enable an Ethernet instance at board level:

```
&ethernet0 {
   status = "okay";                                /* enable ethernet0 */
   pinctrl-0 = <&ethernet0_rmii_pins_a>;           /* configure pinctrl modes for ethernet0
*/
   pinctrl-1 = <&ethernet0_rmii_pins_sleep_a>;     /* configure ethernet0_rmii_pins_sleep_a
as sleep pinctrl configuration for ethernet0 */
   pinctrl-names = "default", "sleep";
   phy-mode = "rmii";                              /* configure ethernet phy mode for
ethernet0 */
   max-speed = <100>;                              /* configure ethernet max speed for
ethernet0 */
   phy-handle = <&phy0>;

   mdio0 {
       #address-cells = <1>;
       #size-cells = <0>;
       compatible = "snps,dwmac-mdio";
       phy0: ethernet-phy@1 {
           reg = <1>;                              /* configure ethernet phy @ for ethernet0
*/
       };
   };
};
```

How to configure Ethernet for :

3.3.1 RMII with Crystal on PHY (Reference clock (standard RMII clock name) is provided by a Phy Crystal)

```
   ethernet0: ethernet@5800a000 {
       compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
       reg = <0x5800a000 0x2000>;
       reg-names = "stmmaceth";
       interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                             <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                             <&exti 70 1>;
       interrupt-names = "macirq",
                         "eth_wake_irq",
```

```
                        "stm32_pwr_wakeup";
        clock-names = "stmmaceth",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
        snps,pbl = <2>;
        snps,en-tx-lpi-clockgating;
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

```
    &ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rmii_pins_a>;
        pinctrl-1 = <&ethernet0_rmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rmii";
        max-speed = <100>;
        phy-handle = <&phy0>;
        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
    };
```

3.3.2 RMII with 25MHz on ETH_CLK (no PHY Crystal), REF_CLK from PHY (Reference clock (standard RMII clock name) is provided by a RCC SoC internal clock)

```
    ethernet0: ethernet@5800a000 {
        compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
        reg = <0x5800a000 0x2000>;
        reg-names = "stmmaceth";
        interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                              <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                              <&exti 70 1>;
        interrupt-names = "macirq",
                          "eth_wake_irq",
                          "stm32_pwr_wakeup";
        clock-names = "stmmaceth",
                      "eth-ck",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHCK_K>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
```

```
        snps,pbl = <2>;
        snps,en-tx-lpi-clockgating;
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

```
   &ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rmii_pins_a>;
        pinctrl-1 = <&ethernet0_rmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rmii";
        max-speed = <100>;
        phy-handle = <&phy0>;
        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
   };
```

+ update stm32mp15-pinctrl.dtsi [4] to add ETHCK pin in ethernet0_rmii_pins_* node:
For example:

```
 <STM32_PINMUX('G', 8, AF2)>, /* ETH_RMII_ETHCK */
```

+ Need also to update TFA devicetree to generate 25Mhz clock (from PLL4P or PLL3Q):
for example if PLL4P in ed1 board: update fdts/stm32mp15xx-edx.dtsi

```
st,pkcs = <
 CLK_CKPER_HSE
 CLK_FMC_ACLK
 CLK_QSPI_ACLK
 - CLK_ETH_DISABLED
 + CLK_ETH_PLL4P
...
 /* VCO = 600.0 MHz => P = 25, Q = 50, R = 50 */
   pll4: st,pll@3 {
     compatible = "st,stm32mp1-pll";
     reg = <3>;
     cfg = < 1 49 23 11 11 PQR(1,1,1) >;
 };
```

3.3.3 RMII with 50MHz on ETH_CLK (no PHY Crystal), internal REF_CLK from RCC (Reference clock (standard RMII clock name) is provided by a RCC SoC internal clock)

```
   ethernet0: ethernet@5800a000 {
        compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
        reg = <0x5800a000 0x2000>;
        reg-names = "stmmaceth";
        interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                              <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
```

```
                              <&exti 70 1>;
        interrupt-names = "macirq",
                          "eth_wake_irq",
                          "stm32_pwr_wakeup";
        clock-names = "stmmaceth",
                      "eth-ck",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHCK_K>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
        snps,pbl = <2>;
        snps,en-tx-lpi-clockgating;
        st,eth_ref_clk_sel;                /* In case of U-Boot */
       or
        st,eth-ref-clk-sel;                /* In case of Linux Kernel */
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

```
   &ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rmii_pins_a>;
        pinctrl-1 = <&ethernet0_rmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rmii";
        max-speed = <100>;
        phy-handle = <&phy0>;
        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
   };
```

+ update stm32mp15-pinctrl.dtsi to add ETHCK pin in ethernet0_rmii_pins_* node:
For example:

```
 <STM32_PINMUX('G', 8, AF2)>, /* ETH_RMII_ETHCK */
```

+ Need also to update TFA to generate 50Mhz clock (from PLL4P or PLL3Q):
for example if PLL4P in ed1 board: update fdts/stm32mp15xx-edx.dtsi

```
st,pkcs = <
 CLK_CKPER_HSE
 CLK_FMC_ACLK
 CLK_QSPI_ACLK
 - CLK_ETH_DISABLED
 + CLK_ETH_PLL4P
...
```

```
/* VCO = 508.0 MHz => P = 50, Q = 60, R = 60 */
  pll4: st,pll@3 {
    compatible = "st,stm32mp1-pll";
    reg = <3>;
   cfg = < 1 49 11 9 9 PQR(1,1,1) >;
 };
```

3.3.4 RGMII with Crystal on PHY, CLK125 from PHY (Reference clock (standard RGMII clock name) is provided by a Phy Crystal)

```
  ethernet0: ethernet@5800a000 {
      compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
      reg = <0x5800a000 0x2000>;
      reg-names = "stmmaceth";
      interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                            <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                            <&exti 70 1>;
      interrupt-names = "macirq",
                        "eth_wake_irq",
                        "stm32_pwr_wakeup";
      clock-names = "stmmaceth",
                    "mac-clk-tx",
                    "mac-clk-rx",
                    "ethstp";
      clocks = <&rcc ETHMAC>,
               <&rcc ETHTX>,
               <&rcc ETHRX>,
               <&rcc ETHSTP>;
      st,syscon = <&syscfg 0x4>;
      snps,mixed-burst;
      snps,pbl = <2>;
      snps,en-tx-lpi-clockgating;
      snps,axi-config = <&stmmac_axi_config_0>;
      snps,tso;
      power-domains = <&pd_core>;
      status = "disabled";
  };
```

```
  &ethernet0 {
      status = "okay";
      pinctrl-0 = <&ethernet0_rgmii_pins_a>;
      pinctrl-1 = <&ethernet0_rgmii_pins_sleep_a>;
      pinctrl-names = "default", "sleep";
      phy-mode = "rgmii";
      max-speed = <1000>;
      phy-handle = <&phy0>;
      mdio0 {
              #address-cells = <1>;
              #size-cells = <0>;
              compatible = "snps,dwmac-mdio";
              phy0: ethernet-phy@0 {
                      reg = <0>;
              };
          };
  };
```

3.3.5 RGMII with 25MHz on ETH_CLK (no PHY Crystal), CLK125 from PHY (Reference clock (standard RGMII clock name) is provided by a RCC SoC internal clock)

```
    ethernet0: ethernet@5800a000 {
        compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
        reg = <0x5800a000 0x2000>;
        reg-names = "stmmaceth";
        interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                              <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                              <&exti 70 1>;
        interrupt-names = "macirq",
                          "eth_wake_irq",
                          "stm32_pwr_wakeup";
        clock-names = "stmmaceth",
                      "eth-ck",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHCK_K>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
        snps,pbl = <2>;
        snps,en-tx-lpi-clockgating;
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

```
    &ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rgmii_pins_a>;
        pinctrl-1 = <&ethernet0_rgmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rgmii";
        max-speed = <1000>;
        phy-handle = <&phy0>;
        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
    };
```

+ update stm32mp15-pinctrl.dtsi to add ETHCK pin in ethernet0_rgmii_pins_* node:
For example:

```
 <STM32_PINMUX('G', 8, AF2)>, /* ETH_RGMII_ETHCK */
```

+ Need also to update TFA to generate 25Mhz clock (from PLL4P or PLL3Q):
for example if PLL4P in ed1 board: update fdts/stm32mp15xx-edx.dtsi

```
st,pkcs = <
 CLK_CKPER_HSE
 CLK_FMC_ACLK
 CLK_QSPI_ACLK
 - CLK_ETH_DISABLED
 + CLK_ETH_PLL4P
...
 /* VCO = 600.0 MHz => P = 25, Q = 50, R = 50 */
   pll4: st,pll@3 {
     compatible = "st,stm32mp1-pll";
     reg = <3>;
     cfg = < 1 49 23 11 11 PQR(1,1,1) >;
 };
```

3.3.6 RGMII with Crystal on PHY, no 125Mhz from PHY

```
   ethernet0: ethernet@5800a000 {
       compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
       reg = <0x5800a000 0x2000>;
       reg-names = "stmmaceth";
       interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                             <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                             <&exti 70 1>;
       interrupt-names = "macirq",
                         "eth_wake_irq",
                         "stm32_pwr_wakeup";
       clock-names = "stmmaceth",
                     "eth-ck",
                     "mac-clk-tx",
                     "mac-clk-rx",
                     "ethstp";
       clocks = <&rcc ETHMAC>,
                <&rcc ETHCK_K>,
                <&rcc ETHTX>,
                <&rcc ETHRX>,
                <&rcc ETHSTP>;
       st,syscon = <&syscfg 0x4>;
       snps,mixed-burst;
       snps,pbl = <2>;
       snps,en-tx-lpi-clockgating;
       st,eth_clk_sel;                /* In case of U-Boot */
      or
       st,eth-clk-sel;                /* In case of Linux Kernel */
       snps,axi-config = <&stmmac_axi_config_0>;
       snps,tso;
       power-domains = <&pd_core>;
       status = "disabled";
};
```

```
   &ethernet0 {
       status = "okay";
       pinctrl-0 = <&ethernet0_rgmii_pins_a>;
       pinctrl-1 = <&ethernet0_rgmii_pins_sleep_a>;
       pinctrl-names = "default", "sleep";
       phy-mode = "rgmii";
       max-speed = <1000>;
       phy-handle = <&phy0>;
       mdio0 {
               #address-cells = <1>;
               #size-cells = <0>;
```

```
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
    };
```

+ update stm32mp15-pinctrl.dtsi to delete CLK125 pin (also no need of ETHCK pin) in ethernet0_rgmii_pins_* node:

+ Need also to update TFA to generate 125Mhz clock (from PLL4P or PLL3Q):
for example if PLL4P in ed1 board: update fdts/stm32mp15xx-edx.dtsi

```
st,pkcs = <
 CLK_CKPER_HSE
 CLK_FMC_ACLK
 CLK_QSPI_ACLK
 - CLK_ETH_DISABLED
 + CLK_ETH_PLL4P
...
 /* VCO = 750.0 MHz => P = 125, Q = 62.5, R = 62.5 */
 pll4: st,pll@3 {
     compatible = "st,stm32mp1-pll";
     reg = <3>;
     cfg = < 3 124 5 11 11 PQR(1,1,1) >;
 };
```

4 How to configure Ethernet using CubeMX

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/net/stmmac.txt
- Documentation/devicetree/bindings/net/stm32-dwmac.txt
- arch/arm/boot/dts/stm32mp151.dtsi , STM32MP151 device tree file
- arch/arm/boot/dts/stm32mp15-pinctrl.dtsi , STM32MP15 pinctrl device tree file

Linux® is a registered trademark of Linus Torvalds.

Operating System

Device Tree

Ethernet

Generic Interrupt Controller

Serial Peripheral Interface

Reset and Clock Control

High Speed External oscillator (STM32 clock source)

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

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

A quality version of this page, approved on *19 March 2021*, was based off this revision.

Contents

1 Purpose

The objective of this chapter is to give general information about the device tree.
An extract of the **device tree specification**[1] explains it as follows:

"A device tree is a tree data structure with nodes that describe the devices in a system. Each node has property/value pairs that describe the characteristics of the device being represented. Each node has exactly one parent except for the root node, which has no parent. ... Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time."

In other words, a device tree describes the hardware that can not be located by probing. For more information, please refer to the device tree specification[1]

1.1 Source files

- **.dts:** The device tree source (DTS). This format is a textual representation of a device tree in a form that can be processed by DTC (Device Tree Compiler) into a binary device tree in the form expected by software components: Linux® Kernel, U-Boot and TF-A.

- **.dtsi:** Source files that can be included from a DTS file.

1.2 Bindings

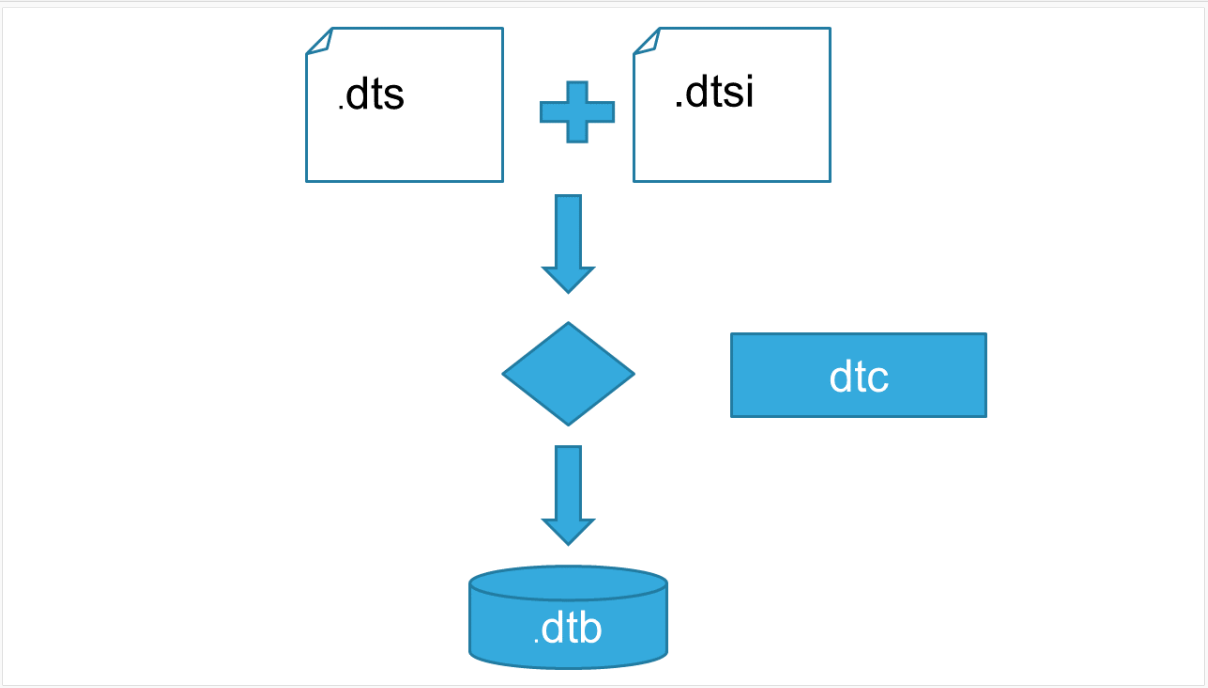
The device tree data structures and properties are named **bindings**. Those bindings are described in:

- The Device tree specification[1] for generic bindings.
- The software component documentations:
  - Linux® Kernel: Linux kernel device tree bindings
  - U-Boot: doc/device-tree-bindings/
  - TF-A: TF-A device tree bindings

1.3 Build

- A tool named DTC (Device Tree Compiler) allows compiling the DTS sources into a binary.
- input file: the *.dts* file described in section above.
- output file: the *.dtb* file described in section above.
- More information are available in DTC manual[2].

- DTC source code is located here[3]. DTC tool is also available directly in particular software components: **Linux Kernel, U-Boot, TF-A ...**. For those components, the device tree building is directly integrated in the component build process.



Device tree build process

If dts files use some defines, dts files should be preprocessed before being compiled by DTC.

1.4 Tools

The device tree compiler offers also some tools:

- **fdtdump:** Print a readable version of a flattened device tree file (dtb)
- **fdtget:** Read properties from a device tree
- **fdtput:** Write properties to a device tree
- ...

There are several ways to get those tools:

- In the device tree compiler project source code[3]
- Directly in software components: **Kernel, u-boot, tf-a ...**
- Available in Debian package[4]

2 STM32

For STM32MP1, the device tree is used by three software components: Linux® kernel, U-Boot and TF-A.

The device tree is part of the OpenSTLinux distribution. It can also be generated by STM32CubeMX tool.

To have more information about the device tree usage on STM32MP1 (how the device tree source files are split, how to find the device tree source files per software components, how is STM32CubeMX generating the device tree ...) see STM32MP15 device tree page.

3 How to go further

- Device Tree for STM32MP [5]
- Device Tree Reference[6] - eLinux.org
- Device Tree usage[7] - eLinux.org

4 References

- 1.01.11.2 https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.2(latest) ,Device tree specification
- https://git.kernel.org/pub/scm/utils/dtc/dtc.git/tree/Documentation/manual.txt(master) ,DTC manual
- 3.03.1 https://git.kernel.org/pub/scm/utils/dtc/dtc.git(master) ,DTC source code
- https://packages.debian.org/search?keywords=device-tree-compiler(master) ,DTC debian package
- https://www.youtube.com/watch?v=a9CZ1Uk3OYQ, Device Tree for STM32MP
- Device Tree Reference, eLinux.org
- Device Tree Usage, eLinux.org

Device Tree Source (in software context) or Digital Temperature Sensor (in peripheral context)

Linux® is a registered trademark of Linus Torvalds.

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Trusted Firmware for Arm® Cortex®-A

Stable: 04.02.2020 - 16:08 / Revision: 04.02.2020 - 16:00

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

Contents

1 Article purpose

The purpose of this article is to:

- briefly introduce the Ethernet peripheral and its main features
- indicate the level of security supported by this hardware block
- explain how each instance can be allocated to the two runtime contexts and linked to the corresponding software components
- explain, when necessary, how to configure the Ethernet peripheral.

2 Peripheral overview

The Ethernet peripheral (ETH) is based on Synopsys DesignWare® Ethernet GMAC IP, which enables the host to communicate data using the Gigabit Ethernet protocol (IEEE 802.3) at 10, 100 and 1000 Mbps.
The peripheral is composed of three main layers: the gigabit ethernet media access controller (GMAC), the MAC transaction layer (MTL), and the MAC DMA controller (MDC). The driver used to drive the ETH is Stmmac.

2.1 Features

The Ethernet peripheral main features are the following:

- Compliance with IEEE 802.3 specifications
- Support for IEEE 1588-2002 and IEEE 1588-2008 standards for precision networked clock synchronization

  IEEE 802.3-az for Energy Efficient Ethernet (EEE)

  IEEE 802.3x flow control automatic transmission of zero-quanta pause frame on flow control input de-assertion.

  IEEE 802.1Q VLAN tag detection for reception frames

  AMBA 2.0 for AHB Master/Slave ports and AMBA 3.0 for AXI Master/Slave ports
- Configurability allowing to support data transfer rates of 10/100/100 Mbps, 10/100 Mbps only or 1000 Mbps only
- Support for multiple TCP/IP offload functions

Refer to STM32MP15 reference manuals for the complete features list, and to the software components, introduced below, to see which features are implemented.

2.2 Security support

The ETH is a **non-secure** peripheral.

3 Peripheral usage and associated software

3.1 Boot time

The Ethernet peripheral can be used at boot time by SSBL (by UBoot with tftp protocol for image loading). See How_to_boot_the_kernel_via_TFTP_from_U-Boot for more details.

3.2 Runtime

3.2.1 Overview

The Ethernet peripheral can be allocated to the Arm®Cortex®-A7 non-secure core to be controlled in Linux® by the NetDev Framework.

3.2.2 Software frameworks

Domain	Peripheral	Software components		Comment
OP-TEE	Linux	STM32Cube		
Networking	ETH		Linux netdev/ethernet framework	

3.2.3 Peripheral configuration

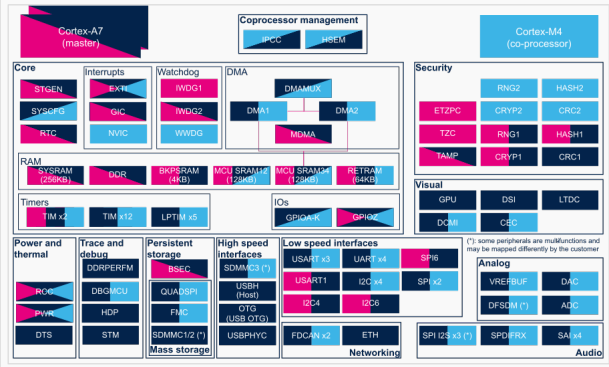
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in the corresponding software framework article. When the Ethernet peripheral is assigned to the Linux®OS, it is configured through the device tree according to the information given in the Ethernet device tree configuration article.

3.2.4 Peripheral assignment

Check boxes illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- means that the peripheral can be assigned () to the given runtime context.
- is used for system peripherals that cannot be unchecked because they are statically connected in the device.

Refer to How to assign an internal peripheral to a runtime context for more information on how to assign peripherals manually or via STM32CubeMX.
The present chapter describes STMicroelectronics recommendations or choice of implementation. Additional possiblities might be described in STM32MP15 reference manuals .



Internal peripherals

Domain	Peripheral	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Networking	ETH	ETH				Assignment (single choice)

4 How to go further

Use this paragraph to add more information and introduce other documentation such as Application Notes (AN)

5 References

Ethernet

Direct Memory Access

Virtual LAN. Network of computers that behave as if they are connected to the same wire even though they may actually be physically located on different segments of a LAN

Advanced High-performance Bus

Second Stage Boot Loader

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Open Portable Trusted Execution Environment

Operating System

Stable: 06.10.2020 - 13:06 / Revision: 06.10.2020 - 12:35

A quality version of this page, approved on *6 October 2020*, was based off this revision.

Contents

1 Article purpose

This article explains how to configure the Ethernet when it is assigned to the Linux®OS. In that case, it is controlled by the Ethernet framework

The configuration is performed using the device tree mechanism that provides a hardware description of the Ethernet peripheral, used by the STM32 DWMAC driver

2 DT bindings documentation

The *Ethernet* is a multifunction device.

Each function is represented by a separate binding document:

- "Generic" Ethernet device tree bindings [1]
- specific STM32 ETH device tree bindings[2]

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.

3.1 DT configuration (STM32 level)

Ethernet peripheral nodes are located in stm32mp151.dtsi [3] file with a disabled status and some required properties such as:

- Physical base address and size of the device register map
- STMMAC interrupts
- stmmaceth clock and Rx, Tx clocks

This is a set of properties that may not vary for a given STM32MP device, such as: register addresses, interrupts, clocks, ...

```
ethernet0: ethernet@5800a000 {
        compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
        reg = <0x5800a000 0x2000>;
        reg-names = "stmmaceth";
        interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_NONE>;
        interrupt-names = "macirq";
        clock-names = "stmmaceth",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
        snps,pbl = <2>;
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

The required and optional properties are fully described in the bindings files.

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

3.2 Ethernet DT configuration (board level)

The device tree board file (.dts) contains all hardware configurations related to board design. The DT node (**"ethernet"**) should be updated to:

- Enable the Ethernet block by setting **status = "okay".**
- Configure the pins in use via pinctrl, through **pinctrl-0** (default pins), **pinctrl-1** (sleep pins) and **pinctrl-names**.
- Configure Ethernet interface used **phy-mode = "rgmii".**, (rmii, mii, gmii).
- Configure Ethernet max speed **max-speed = <1000>".**.

```
&ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rgmii_pins_a>;
        pinctrl-1 = <&ethernet0_rgmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rgmii";
        max-speed = <1000>;
        phy-handle = <&phy0>;

        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@1 {
                        reg = <1>;
                };
        };
};
```

3.3 DT configuration examples

The example below shows how to configure and enable an Ethernet instance at board level:

```
&ethernet0 {
   status = "okay";                                /* enable ethernet0 */
   pinctrl-0 = <&ethernet0_rmii_pins_a>;           /* configure pinctrl modes for ethernet0
*/
   pinctrl-1 = <&ethernet0_rmii_pins_sleep_a>;     /* configure ethernet0_rmii_pins_sleep_a
as sleep pinctrl configuration for ethernet0 */
   pinctrl-names = "default", "sleep";
   phy-mode = "rmii";                              /* configure ethernet phy mode for
ethernet0 */
   max-speed = <100>;                              /* configure ethernet max speed for
ethernet0 */
   phy-handle = <&phy0>;

   mdio0 {
       #address-cells = <1>;
       #size-cells = <0>;
       compatible = "snps,dwmac-mdio";
       phy0: ethernet-phy@1 {
           reg = <1>;                              /* configure ethernet phy @ for ethernet0
*/
       };
   };
};
```

How to configure Ethernet for :

3.3.1 RMII with Crystal on PHY (Reference clock (standard RMII clock name) is provided by a Phy Crystal)

```
   ethernet0: ethernet@5800a000 {
       compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
       reg = <0x5800a000 0x2000>;
       reg-names = "stmmaceth";
       interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                             <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                             <&exti 70 1>;
       interrupt-names = "macirq",
                         "eth_wake_irq",
```

```
                        "stm32_pwr_wakeup";
        clock-names = "stmmaceth",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
        snps,pbl = <2>;
        snps,en-tx-lpi-clockgating;
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

```
    &ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rmii_pins_a>;
        pinctrl-1 = <&ethernet0_rmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rmii";
        max-speed = <100>;
        phy-handle = <&phy0>;
        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
    };
```

3.3.2 RMII with 25MHz on ETH_CLK (no PHY Crystal), REF_CLK from PHY (Reference clock (standard RMII clock name) is provided by a RCC SoC internal clock)

```
    ethernet0: ethernet@5800a000 {
        compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
        reg = <0x5800a000 0x2000>;
        reg-names = "stmmaceth";
        interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                              <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                              <&exti 70 1>;
        interrupt-names = "macirq",
                          "eth_wake_irq",
                          "stm32_pwr_wakeup";
        clock-names = "stmmaceth",
                      "eth-ck",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHCK_K>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
```

```
        snps,pbl = <2>;
        snps,en-tx-lpi-clockgating;
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

```
   &ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rmii_pins_a>;
        pinctrl-1 = <&ethernet0_rmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rmii";
        max-speed = <100>;
        phy-handle = <&phy0>;
        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
   };
```

+ update stm32mp15-pinctrl.dtsi [4] to add ETHCK pin in ethernet0_rmii_pins_* node:
For example:

```
 <STM32_PINMUX('G', 8, AF2)>, /* ETH_RMII_ETHCK */
```

+ Need also to update TFA devicetree to generate 25Mhz clock (from PLL4P or PLL3Q):
for example if PLL4P in ed1 board: update fdts/stm32mp15xx-edx.dtsi

```
st,pkcs = <
 CLK_CKPER_HSE
 CLK_FMC_ACLK
 CLK_QSPI_ACLK
 - CLK_ETH_DISABLED
 + CLK_ETH_PLL4P
...
 /* VCO = 600.0 MHz => P = 25, Q = 50, R = 50 */
   pll4: st,pll@3 {
     compatible = "st,stm32mp1-pll";
     reg = <3>;
     cfg = < 1 49 23 11 11 PQR(1,1,1) >;
 };
```

3.3.3 RMII with 50MHz on ETH_CLK (no PHY Crystal), internal REF_CLK from RCC (Reference clock (standard RMII clock name) is provided by a RCC SoC internal clock)

```
   ethernet0: ethernet@5800a000 {
        compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
        reg = <0x5800a000 0x2000>;
        reg-names = "stmmaceth";
        interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                              <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
```

```
                              <&exti 70 1>;
        interrupt-names = "macirq",
                          "eth_wake_irq",
                          "stm32_pwr_wakeup";
        clock-names = "stmmaceth",
                      "eth-ck",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHCK_K>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
        snps,pbl = <2>;
        snps,en-tx-lpi-clockgating;
        st,eth_ref_clk_sel;                /* In case of U-Boot */
       or
        st,eth-ref-clk-sel;                /* In case of Linux Kernel */
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

```
   &ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rmii_pins_a>;
        pinctrl-1 = <&ethernet0_rmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rmii";
        max-speed = <100>;
        phy-handle = <&phy0>;
        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
   };
```

+ update stm32mp15-pinctrl.dtsi to add ETHCK pin in ethernet0_rmii_pins_* node:
For example:

```
 <STM32_PINMUX('G', 8, AF2)>, /* ETH_RMII_ETHCK */
```

+ Need also to update TFA to generate 50Mhz clock (from PLL4P or PLL3Q):
for example if PLL4P in ed1 board: update fdts/stm32mp15xx-edx.dtsi

```
st,pkcs = <
 CLK_CKPER_HSE
 CLK_FMC_ACLK
 CLK_QSPI_ACLK
 - CLK_ETH_DISABLED
 + CLK_ETH_PLL4P
...
```

```
/* VCO = 508.0 MHz => P = 50, Q = 60, R = 60 */
  pll4: st,pll@3 {
    compatible = "st,stm32mp1-pll";
    reg = <3>;
   cfg = < 1 49 11 9 9 PQR(1,1,1) >;
 };
```

3.3.4 RGMII with Crystal on PHY, CLK125 from PHY (Reference clock (standard RGMII clock name) is provided by a Phy Crystal)

```
  ethernet0: ethernet@5800a000 {
      compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
      reg = <0x5800a000 0x2000>;
      reg-names = "stmmaceth";
      interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                            <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                            <&exti 70 1>;
      interrupt-names = "macirq",
                        "eth_wake_irq",
                        "stm32_pwr_wakeup";
      clock-names = "stmmaceth",
                    "mac-clk-tx",
                    "mac-clk-rx",
                    "ethstp";
      clocks = <&rcc ETHMAC>,
               <&rcc ETHTX>,
               <&rcc ETHRX>,
               <&rcc ETHSTP>;
      st,syscon = <&syscfg 0x4>;
      snps,mixed-burst;
      snps,pbl = <2>;
      snps,en-tx-lpi-clockgating;
      snps,axi-config = <&stmmac_axi_config_0>;
      snps,tso;
      power-domains = <&pd_core>;
      status = "disabled";
  };
```

```
  &ethernet0 {
      status = "okay";
      pinctrl-0 = <&ethernet0_rgmii_pins_a>;
      pinctrl-1 = <&ethernet0_rgmii_pins_sleep_a>;
      pinctrl-names = "default", "sleep";
      phy-mode = "rgmii";
      max-speed = <1000>;
      phy-handle = <&phy0>;
      mdio0 {
              #address-cells = <1>;
              #size-cells = <0>;
              compatible = "snps,dwmac-mdio";
              phy0: ethernet-phy@0 {
                      reg = <0>;
              };
          };
  };
```

3.3.5 RGMII with 25MHz on ETH_CLK (no PHY Crystal), CLK125 from PHY (Reference clock (standard RGMII clock name) is provided by a RCC SoC internal clock)

```
    ethernet0: ethernet@5800a000 {
        compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
        reg = <0x5800a000 0x2000>;
        reg-names = "stmmaceth";
        interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                              <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                              <&exti 70 1>;
        interrupt-names = "macirq",
                          "eth_wake_irq",
                          "stm32_pwr_wakeup";
        clock-names = "stmmaceth",
                      "eth-ck",
                      "mac-clk-tx",
                      "mac-clk-rx",
                      "ethstp";
        clocks = <&rcc ETHMAC>,
                 <&rcc ETHCK_K>,
                 <&rcc ETHTX>,
                 <&rcc ETHRX>,
                 <&rcc ETHSTP>;
        st,syscon = <&syscfg 0x4>;
        snps,mixed-burst;
        snps,pbl = <2>;
        snps,en-tx-lpi-clockgating;
        snps,axi-config = <&stmmac_axi_config_0>;
        snps,tso;
        power-domains = <&pd_core>;
        status = "disabled";
};
```

```
    &ethernet0 {
        status = "okay";
        pinctrl-0 = <&ethernet0_rgmii_pins_a>;
        pinctrl-1 = <&ethernet0_rgmii_pins_sleep_a>;
        pinctrl-names = "default", "sleep";
        phy-mode = "rgmii";
        max-speed = <1000>;
        phy-handle = <&phy0>;
        mdio0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
    };
```

+ update stm32mp15-pinctrl.dtsi to add ETHCK pin in ethernet0_rgmii_pins_* node:
For example:

```
 <STM32_PINMUX('G', 8, AF2)>, /* ETH_RGMII_ETHCK */
```

+ Need also to update TFA to generate 25Mhz clock (from PLL4P or PLL3Q):
for example if PLL4P in ed1 board: update fdts/stm32mp15xx-edx.dtsi

```
st,pkcs = <
 CLK_CKPER_HSE
 CLK_FMC_ACLK
 CLK_QSPI_ACLK
 - CLK_ETH_DISABLED
 + CLK_ETH_PLL4P
...
 /* VCO = 600.0 MHz => P = 25, Q = 50, R = 50 */
   pll4: st,pll@3 {
     compatible = "st,stm32mp1-pll";
     reg = <3>;
     cfg = < 1 49 23 11 11 PQR(1,1,1) >;
 };
```

3.3.6 RGMII with Crystal on PHY, no 125Mhz from PHY

```
   ethernet0: ethernet@5800a000 {
       compatible = "st,stm32mp1-dwmac", "snps,dwmac-4.20a";
       reg = <0x5800a000 0x2000>;
       reg-names = "stmmaceth";
       interrupts-extended = <&intc GIC_SPI 61 IRQ_TYPE_LEVEL_HIGH>,
                             <&intc GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>,
                             <&exti 70 1>;
       interrupt-names = "macirq",
                         "eth_wake_irq",
                         "stm32_pwr_wakeup";
       clock-names = "stmmaceth",
                     "eth-ck",
                     "mac-clk-tx",
                     "mac-clk-rx",
                     "ethstp";
       clocks = <&rcc ETHMAC>,
                <&rcc ETHCK_K>,
                <&rcc ETHTX>,
                <&rcc ETHRX>,
                <&rcc ETHSTP>;
       st,syscon = <&syscfg 0x4>;
       snps,mixed-burst;
       snps,pbl = <2>;
       snps,en-tx-lpi-clockgating;
       st,eth_clk_sel;                /* In case of U-Boot */
      or
       st,eth-clk-sel;                /* In case of Linux Kernel */
       snps,axi-config = <&stmmac_axi_config_0>;
       snps,tso;
       power-domains = <&pd_core>;
       status = "disabled";
};
```

```
   &ethernet0 {
       status = "okay";
       pinctrl-0 = <&ethernet0_rgmii_pins_a>;
       pinctrl-1 = <&ethernet0_rgmii_pins_sleep_a>;
       pinctrl-names = "default", "sleep";
       phy-mode = "rgmii";
       max-speed = <1000>;
       phy-handle = <&phy0>;
       mdio0 {
               #address-cells = <1>;
               #size-cells = <0>;
```

```
                compatible = "snps,dwmac-mdio";
                phy0: ethernet-phy@0 {
                        reg = <0>;
                };
        };
    };
```

+ update stm32mp15-pinctrl.dtsi to delete CLK125 pin (also no need of ETHCK pin) in ethernet0_rgmii_pins_* node:

+ Need also to update TFA to generate 125Mhz clock (from PLL4P or PLL3Q):
for example if PLL4P in ed1 board: update fdts/stm32mp15xx-edx.dtsi

```
st,pkcs = <
 CLK_CKPER_HSE
 CLK_FMC_ACLK
 CLK_QSPI_ACLK
 - CLK_ETH_DISABLED
 + CLK_ETH_PLL4P
...
 /* VCO = 750.0 MHz => P = 125, Q = 62.5, R = 62.5 */
 pll4: st,pll@3 {
     compatible = "st,stm32mp1-pll";
     reg = <3>;
     cfg = < 3 124 5 11 11 PQR(1,1,1) >;
 };
```

4 How to configure Ethernet using CubeMX

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/net/stmmac.txt
- Documentation/devicetree/bindings/net/stm32-dwmac.txt
- arch/arm/boot/dts/stm32mp151.dtsi , STM32MP151 device tree file
- arch/arm/boot/dts/stm32mp15-pinctrl.dtsi , STM32MP15 pinctrl device tree file

Operating System

Device Tree

Ethernet

Generic Interrupt Controller

Serial Peripheral Interface

Reset and Clock Control

High Speed External oscillator (STM32 clock source)

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Stable: 19.10.2020 - 12:02 / Revision: 19.10.2020 - 12:01

A quality version of this page, approved on *19 October 2020*, was based off this revision.

This article gives information about the Linux® Ethernet framework, provides its composition and explains how to configure and use it.

Contents

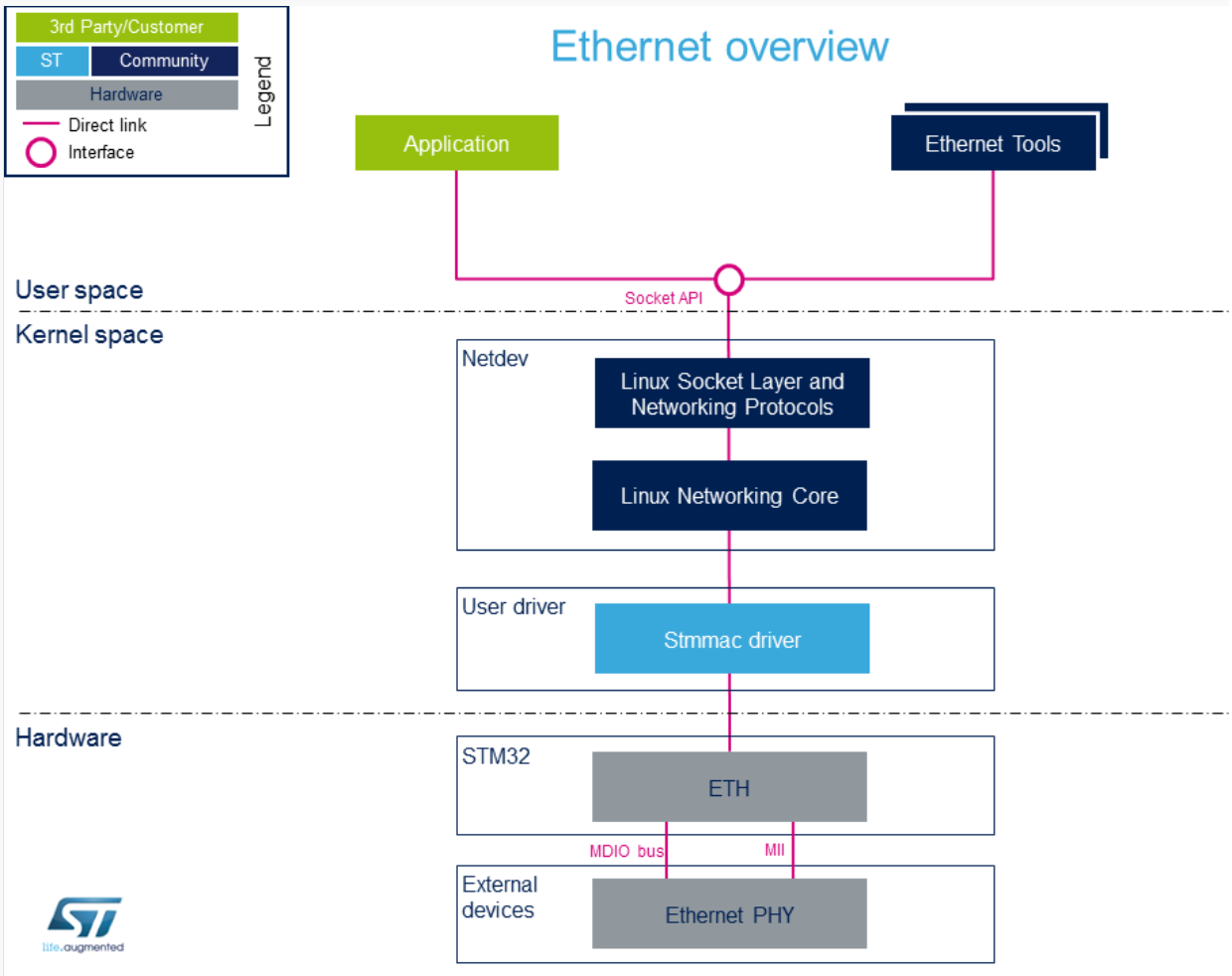
1 Framework purpose

Ethernet is a way of connecting devices together in a local area network or LAN. An Ethernet protocol is used to transmit packets of data containing any sort of information. Any two devices that are connected to the network can exchange information through an Ethernet connection. Ethernet provides a fast, efficient, and direct connection to a router.

Ethernet can be used in many different use cases, as mentioned in How to use Ethernet section:

- How to perform remote connection SSH
- How to perform ping test PING

2 System overview



Ethernet Overview

2.1 Component description

From User space to hardware

- **Application** (User space)

There are a lot of applications using ethernet: Internet Browser, Streaming applications, FTP applications etc..

The main interface that is used between an application and the Networking protocols is a socket [1]

- **Ethernet tools** (User space)

A set of utilities is available to manage and maintain networks: ethtool, ping, route, ifconfig etc..

- **Linux Socket Layer and Networking Protocols** (Kernel space)

The socket layer [2] is a uniform interface between the user process and the network protocol [3] stacks within the kernel

- **Linux Networking Core** (Kernel space)

The kernel network layer adapts the message with the transport protocol in use. The network subsystem of the Linux kernel is designed to be completely protocol-independent.

- **Stmmac Driver** (Kernel space)

This is the driver for the MAC 10/100/1000 on-chip Ethernet controllers (Synopsys IP blocks).
Documentation/networking/stmmac.txt[4]

- **ETH** (Hardware)

This is the Ethernet IP: GMAC [5]

- **Ethernet phy** (Hardware)

The Ethernet PHY is connected to a media access controller (MAC). The MAC controls the data-link-layer portion of the OSI model.
The media-independent interface (MII) defines the interface between the MAC and the PHY.
Variations of the MII are available (RGMII, GMII, RMII, MII) that provide minimal pin count and varied data rates depending on system requirements.
The MDIO bus includes two signals:
- MDC clock: driven by the MAC device to the PHY.
- MDIO data: bidirectional, it is driven by the PHY to provide register data at the end of a read operation.
The connector used by ethernet phy is RJ45.

2.2 API description

The Ethernet API is documented in the Linux Kernel[6].

3 Configuration

3.1 Kernel configuration

The Ethernet API is activated by default in ST deliveries. Nevertheless, if a specific configuration is required, one can use Linux Menuconfig tool: Menuconfig or how to configure kernel and select:

For Network features:

```
[*] Networking support  --->
    [*] Networking options  --->
        [*] Packet socket
        [*] TCP/IP networking
          [*] IP: kernel level autoconfiguration
            [*] IP: DHCP support
            [*] IP: BOOTP support
            [*] IP: RARP support
        [*] INET: socket monitoring interface
        [*] The IPv6 protocol
        [*] DNS Resolver support
```

For Phy (Generic PHY support) :

```
[*] Device Drivers  --->
  [*] PHY Subsystem   --->
    [*] PHY Core
```

For STM32 DWMAC :

```
[*] Device Drivers  --->
    [*]   Network device support  --->
        [*]   Ethernet driver support  --->
             [*]   STMicroelectronics devices
                [*]   STMicroelectronics 10/100/1000/EQOS Ethernet driver
                [*]   STMMAC Platform bus support
                   [*]   Generic driver for DWMAC
                   [*]   STM32 DWMAC support
```

3.2 Device tree configuration

DT bindings documentation deals with all required or optional device tree properties.

Detailed DT configuration for STM32 internal peripherals: Ethernet device tree configuration.

4 How to use Ethernet

4.1 How to use the Ethernet user space interface

Please see examples based on the following use cases:

- How to configure ethernet interface: How to configure ethernet interface
- How to perform ssh connection: How to perform ssh connection
- How to perform ping test: How to perform ping test

5 How to trace and debug the framework

5.1 How to monitor

5.1.1 How to monitor with sysfs

sysfs entry can be used to browse for available descriptors and hardware capabilies.

```
Board $> /sys/kernel/debug/stmmaceth/eth0# ls
   descriptors_status  dma_cap
 root@stm32mp1://sys/kernel/debug/stmmaceth/eth0# cat descriptors_status
 RX Queue 0:
 Descriptor ring:
 0 [0xf4e8d000]: 0xecb01842 0x0 0x0 0x81000000
 1 [0xf4e8d010]: 0xecb02042 0x0 0x0 0x81000000
 ....
 root@stm32mp1://sys/kernel/debug/stmmaceth/eth0# cat dma_cap
 ==============================
         DMA HW features
 ==============================
      10/100 Mbps: Y
      1000 Mbps: Y
      Half duplex: Y
      Hash Filter: Y
      Multiple MAC address registers: Y
      PCS (TBI/SGMII/RTBI PHY interfaces): N
      SMA (MDIO) Interface: Y
      PMT Remote wake up: Y
      PMT Magic Frame: Y
      RMON module: Y
      IEEE 1588-2002 Time Stamp: N
      IEEE 1588-2008 Advanced Time Stamp: Y
      802.3az - Energy-Efficient Ethernet (EEE): Y
      AV features: Y
      Checksum Offload in TX: Y
      IP Checksum Offload in RX: Y
      RXFIFO > 2048bytes: N
      Number of Additional RX channel: 1
      Number of Additional TX channel: 2
      Enhanced descriptors: N
```

5.1.2 Other ways of monitoring

Ethtool is a Linux-based utility for displaying and modifying some parameters of the network interface controllers (NICs) and their device drivers.

```
Board $> ethtool eth0
      Settings for eth0:
      Supported ports: [ TP AUI BNC MII FIBRE ]
      Supported link modes:   10baseT/Half 10baseT/Full
                              100baseT/Half 100baseT/Full
                              1000baseT/Half 1000baseT/Full
      Supported pause frame use: Symmetric Receive-only
      Supports auto-negotiation: Yes
      Advertised link modes:  10baseT/Half 10baseT/Full
                              100baseT/Half 100baseT/Full
                              1000baseT/Half 1000baseT/Full
```

```
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Link partner advertised link modes:  10baseT/Half 10baseT/Full
                                             100baseT/Half 100baseT/Full
                                             1000baseT/Full
        Link partner advertised pause frame use: Symmetric
        Link partner advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: ug
        Wake-on: d
        Current message level: 0x0000003f (63)
                               drv probe link timer ifdown ifup
        Link detected: yes
```

5.2 How to trace

The Ethernet Framework (and specifically the stmmac driver) prints out information and error messages in the kernel console. They are available via dmesg command:

```
Board $> dmesg | grep ethernet
[    1.454632] stm32-dwmac 5800a000.ethernet: PTP uses main clock
[    1.459010] stm32-dwmac 5800a000.ethernet: no reset control found
[    1.465199] stm32-dwmac 5800a000.ethernet: No phy clock provided...
[    1.472347] stm32-dwmac 5800a000.ethernet: User ID: 0x40, Synopsys ID: 0x42
[    1.478319] stm32-dwmac 5800a000.ethernet:   DWMAC4/5
[    1.483310] stm32-dwmac 5800a000.ethernet: DMA HW capability register supported
[    1.490564] stm32-dwmac 5800a000.ethernet: RX Checksum Offload Engine supported
[    1.497888] stm32-dwmac 5800a000.ethernet: TX Checksum insertion supported
[    1.504753] stm32-dwmac 5800a000.ethernet: Wake-Up On Lan supported
[    1.510994] stm32-dwmac 5800a000.ethernet: TSO supported
[    1.516329] stm32-dwmac 5800a000.ethernet: TSO feature enabled
[    1.522143] stm32-dwmac 5800a000.ethernet: Enable RX Mitigation via HW Watchdog Timer
[   12.356485] stm32-dwmac 5800a000.ethernet eth0: No Safety Features support found
[   12.426208] stm32-dwmac 5800a000.ethernet eth0: IEEE 1588-2008 Advanced Timestamp
supported
[   12.481051] stm32-dwmac 5800a000.ethernet eth0: registered PTP clock
[   14.951370] stm32-dwmac 5800a000.ethernet eth0: Link is Up - 1Gbps/Full - flow control
rx/tx
```

It is possible to modify the amount of 'debugging messages/data' returned by the Ethernet driver with ethtool. More documentation is available in Documentation/networking/netif-msg.txt[7] in kernel source folder.

Ethtool to set the message level:

```
Board $> ethtool -s eth1 msglvl [level]
```

5.3 How to debug

During Ethernet bring up, there are 2 frequent errors:

- DMA reset error:

```
[   15.650981] dwmac4_dma_reset err
[   15.652849] stm32-dwmac 5800a000.ethernet: Failed to reset the dma
[   15.659006] stm32-dwmac 5800a000.ethernet eth0: stmmac_hw_setup: DMA engine
initialization failed
[   15.668518] stm32-dwmac 5800a000.ethernet eth0: stmmac_open: Hw setup failed
```

When this error occurs, it is linked to the DMA Software Reset (not linked to memory transfert)

Definition of the Software Reset in GMAC specification:

```
When this bit is set, the MAC and the DMA controller reset the logic
and all internal registers of the DMA, MTL, and MAC. This bit is
automatically cleared after the reset operation is complete in all
DWC_ether_qos clock domains. Before reprogramming any DWC_ether_qos
register, a value of zero should be read in this bit.
*Note*: The reset operation is complete only when all resets in all
active clock domains are de-asserted. Therefore, it is essential that
all PHY inputs clocks (applicable for the selected PHY interface) are
present for software reset completion. The time to complete the
software reset operation depends on the frequency of the slowest active clock.
Access restriction applies. Setting 1 sets. Self-cleared. Setting 0
has no effect.
```

- Ethernet clock tree error:

The GMAC IP verifies that the Ethernet clock tree is well configured. When this error occurs, it is due to the Ethernet PHY that do not detect all needed clocks (tx, rx, aclk or hclk).

To solve this issue:

```
 - check that the pinctrl of each clock is well configured
 - check if syscfg register is well configured (in Ethernet clock tree there are some
gating/mux configured with syscfg)
```

6 Source code location

The source files are located inside the Linux kernel.

- **Ethernet driver**: dwmac-stm32.c[8]

7 References

- [1], Berkeley sockets
- [2], Socket Layer
- [3], Internet Protocol
- https://www.kernel.org/doc/Documentation/networking/stmmac.txt, More information
- [4], DesignWare Ethernet GMAC IP
- Linux Networking and Network Devices APIs
- [5], Documentation/networking/netif-msg.txt
- [6],dwmac-stm32.c

Ethernet

Application programming interface

Dynamic Host Configuration Protocol (See https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol for more details)

Device Tree

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Receive

Direct Memory Access

media access control address (https://en.wikipedia.org/wiki/MAC_address)

Transmit

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

A quality version of this page, approved on *11 June 2020*, was based off this revision.

This article explains how the Linux® **pinctrl** framework manages IOs/pins, how to configure it, and how to use it.

Contents

1 Framework purpose

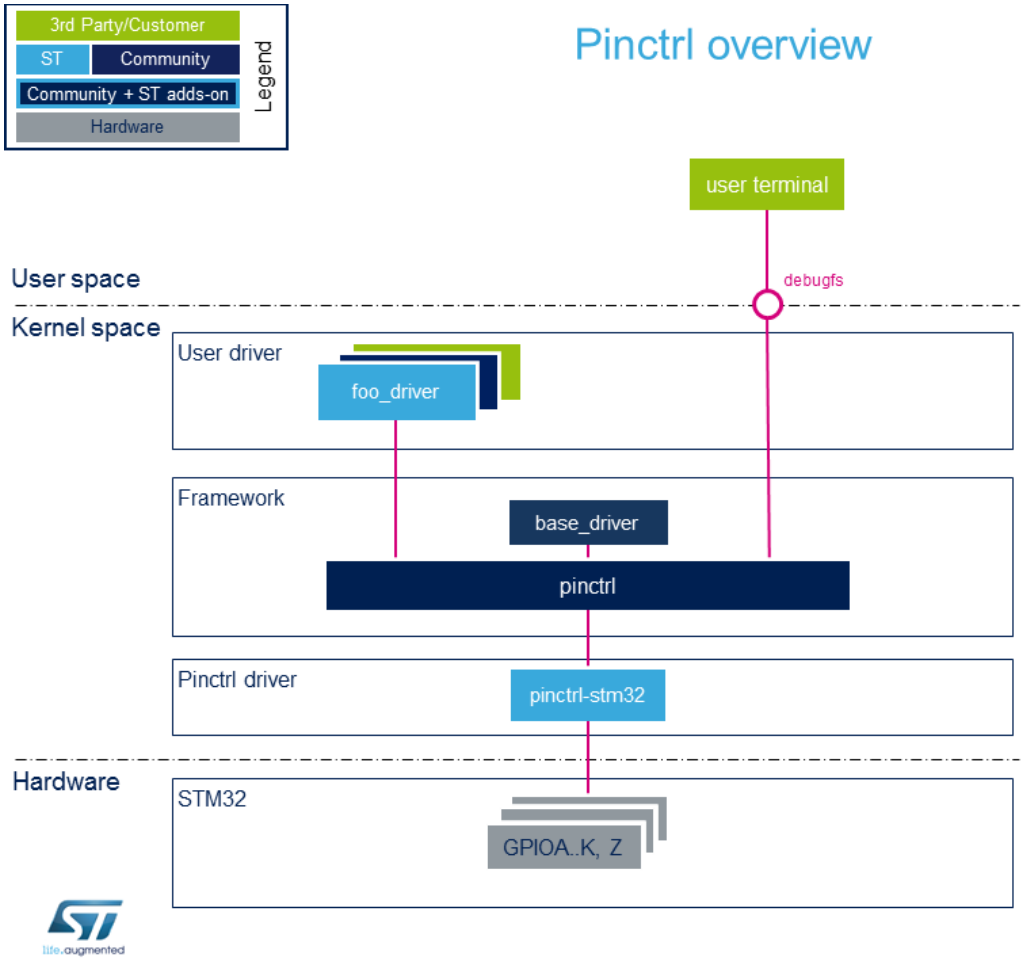
Many of the microprocessor pins (with digital I/O or analog pin type) are multiplexed between different functions: GPIO, alternate function(s).
Pinctrl framework is used to:

- Configure pin hardware settings: multiplexing, pull-up/pull-down, open-drain ...
- Provide information through debugfs

Pinctrl framework is the Linux framework to configure and control the microprocessor pins. There are 2 ways to use it:

- A pin (or group of pins) is controlled by a hardware block, then pinctrl will apply the pin configuration given by the device tree (it just applies devicetree configuration)
- A pin needs to be controlled by software (typically a GPIO), then GPIOLib framework will be used to control this pin on top of pinctrl framework. Refer to GPIOLib overview.

2 System overview



2.1 Component description

- **Pinctrl:** the pinctrl framework **core**, its role is to:
  - provide API to other drivers
  - call specific vendor callback for pin configuration (muxing end setting)
  - create logical pin mapping and guarantee pin exclusivity for a device.
- **Pinctrl-stm32:** microprocessor **specific** pinctrl driver, its role is to:
  - register vendor specific functions (callback) to pinctrl framework
  - access to hardware registers to configure pins (muxing and all pins capabilities)
  - provide other services described in GPIOLib overview.
- **Base driver:** generic kernel driver in charge of getting pin information through the device tree for a device and to register those pins to the pinctrl framework.
- **Foo_driver:**
  - Foo_driver could be any driver that needs specific pins configuration. Note that "default" pins configuration is managed by the kernel base before foo_driver probe. No action is needed by the foo driver.

- this configuration is described in the device tree file. See Pinctrl device tree configuration.

- **debugfs:**
  - provides debug interface available through user terminal, including pin configurations, muxing... See How_to_monitor_with_debugfs.

2.2 API description

- **Kernel space API**: Pinctrl API provides API interface to user driver.
  - Main useful API functions are:

    -*devm_pinctrl_get()*: call to get all pinctrl information.

    -*pinctrl_lookup_state()*: call to obtain a pinctrl state struct from a name.

    -*pinctrl_select_state()*: call to select a pinctrl state struct. After a call to this function, pins are configured.
  - Possible standard state names are: **"default"**, **"init"**, **"sleep"** and **"idle"**.
  - Pinctrl API functions to control those standard states are:

    -*pinctrl_pm_select_sleep_state*: call to select **"sleep"** state defined in device tree.

    -*pinctrl_pm_select_idle_state*: call to select **"idle"** state defined in device tree.

    -*pinctrl_pm_select_default_state*: call to select **"default"** state in device tree
  - See pinctrl kernel documentation[1] for more API function descriptions.

- **debugfs:**

  See How_to_monitor_with_debugfs

3 Configuration

3.1 Kernel configuration

Pinctrl framework and driver are enabled by default.

3.2 Device tree configuration

Refer to Pinctrl device tree configuration.

4 How to use the framework

For a device, there are two ways to use pinctrl framework:

- standard pinctrl utilization
- custom (+standard) pinctrl utilization

4.1 Standard

- To simplify kernel development and avoid code duplication, Linux kernel is in charge to call pinctrl framework to apply pin states (pins configuration). It is possible when standard entries are used in device tree for "pinctrl-names". Possible standard names are: **"default"**, **"init"**, **"sleep"** and **"idle"**.

- **Device tree part:** when using this approach, since Kernel base driver calls pinctrl framework, the user has to write device tree configuration. It means:

  - **Write pin states**: pin states nodes are defined inside the pin controller device node that contains several information about pin configuration. It can be for one pin or a group of pins. This information is not generic and depends on each pin controller driver. See Pinctrl device tree configuration for details.

```
pincontroller {
        foo_state_pins_a {
                "pins configuration: muxing, pull-up/pull-down, ..."
        };

        foo_state_pins_b {
                "pins configuration: muxing, pull-up/pull-down, ..."
        };

        foo_state_pins_sleep_b {
                "pins configuration: muxing, pull-up/pull-down, ..."
        };
}
```

  - **Invoke pin configuration inside user device node**.

```
foo_device {                                    comments
       ...
        pinctrl-names = "default";               -->Standard name known by Linux Kernel.
It's mapped on pinctrl-0 state.
        pinctrl-0 = <&foo_state_pins_a>;         -->Phandle to a pin state node(see above).
       ...
};
```

If needed two pin nodes *foo_state_pins_a* and *foo_state_pins_b* can be used for a same state:

```
foo_device {
        ...
        pinctrl-names = "default";
        pinctrl-0 = <&foo_state_pins_a &foo_state_pins_b>;
        ...
};
```

Two differents states **"default"** and **"sleep"** can also be defined. First name **"default"** is mapped to the first state "pinctrl-0", second name **"sleep"** is mapped to the second state "pinctrl-1":

```
foo_device {
        ...
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&foo_state_pins_a>;
        pinctrl-1 = <&foo_state_pins_sleep_a>;
        ...
};
```

- **Base driver part**[2]

The base driver is in charge to **register** pin states to devices that use standard names as **"default"**, **"idle"**, **"sleep"**, **"init"**. This driver is in charge to **select** **"default"** and **"init"** state:

-If **"default"** state is defined in device tree, this state is selected before the driver probe.

-If **"init"** and **"default"** state are defined, the **"init"** state is selected before the driver probe and the **"default"** state is selected after the driver probe. It is mainly used to avoid glitches.

- **Foo driver part**

As explain above the base driver is in charge to select **"default"** and **"init"** states at probe time. To select **"idle"** and **"sleep"** states, the foo driver has to call pinctrl framework API:

"sleep" and **"idle"** states are mainly used for power management. Indeed to reduce leakage and power consumption, pin settings are changed when the device is not in use. In this case *pinctrl_pm_select_sleep_state* and *pinctrl_pm_select_idle_state* functions can be used. When the device is used again, **"default"** state has to be restored, then *pinctrl_pm_select_default_state* is used.

4.2 Custom

- Sometimes, using standard pin states (managed by base driver and not by concerned foo_driver) is not enough. Foo_driver may need to control pin states at runtime. In such a case it will be up to foo_driver to call framework API.
- The custom pinctrl usage may cohabit with the standard usage explained in previous section.
- Extracted from documentation[1], here is an example on how to use 2 different configurations inside a device driver:
  - **device tree part**

    - **Write pin states**: pin states nodes are defined inside the pin controller device node that contains several information about pin configuration. It can be for one pin or a group of pins. This information is not generic and depends on each pin controller driver. See Pinctrl device tree configuration for details.

```
pincontroller {
        foo_state_pins_a {
                "pins configuration: muxing, pull-up/pull-down, ..."
        };

        foo_state_pins_b {
                "pins configuration: muxing, pull-up/pull-down, ..."
        };
}
```

-Invoke pin configuration inside user device node.

```
       foo_device {
                pinctrl-names = "state-A", "state-B";
                pinctrl-0 = <&state_pins_A>;
                pinctrl-1 = <&state_pins_B>;
        };
```

- foo driver part

  - Initialization part:

```
#include <linux/pinctrl/consumer.h>

struct pinctrl *p;
struct pinctrl_state *s1, *s2;

foo_probe()
{
        /* Setup */
        p = devm_pinctrl_get(&device);
        if (IS_ERR(p))
                ...

        s1 = pinctrl_lookup_state(foo->p, "state-A");
        if (IS_ERR(s1))
                ...

        s2 = pinctrl_lookup_state(foo->p, "state-B");
        if (IS_ERR(s2))
                ...
}
```

- Runtime usage: each state can be selected at runtime.

```
foo_switch()
{
        /* Select pinctrl state A */
        ret = pinctrl_select_state(s1);
        if (ret < 0)
            ...

        ...

        /* select pinctrl state B */
```

```
        ret = pinctrl_select_state(s2);
        if (ret < 0)
            ...

        ...
}
```

- See mmci driver example for a real use case (search for "pinctrl_select_state").

5 How to trace and debug the framework

5.1 How to monitor

5.1.1 How to monitor with debugfs

Some information about pin controller / pins states / pins configurations is available in Debugfs interface. There are two levels of information:

1 Generic information:

```
Board $> ls -l /sys/kernel/debug/pinctrl/
    |
    +---pinctrl-devices      | List of pin ctroller devices.
    +---pinctrl-handles      | List of all pin states registered.
    +---pinctrl-maps         | List of all pin states registered per pin used.
    +---soc:pin-controller-z@54004000    | Folder which contains pins information for a
pin controller.
    +---soc:pin-controller@50002000    | Folder which contains pins information for a pin
controller.
```

2 Pin controller information:

```
Board $> ls -l /sys/kernel/debug/pinctrl/soc:pin-controller@50002000
    |
    +---gpio-ranges          | Provides mapping between logical address space and pins
address space for GPIOs.
    +---pinconf-config       | Provides modified pins at runtime.  Not supported.
    +---pinconf-groups       | Provides pin config settings per pin group.
    +---pinconf-pins         | Provides all pins settings. It reflects the hardware
values.
    +---pingroups            | Provides registered pin groups.
    +---pinmux-functions     | Provides all possibles muxing available.
    +---pinmux-pins          | Provides a list for each pin the muxing selected and the
device which use the pin.
    +---pins                 | Provides list of all pins.
```

5.2 How to trace

- The following extract of kernel log shows that pin controller is well probed:

```
[    0.353613] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOA bank added
[    0.360539] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOB bank added
[    0.367344] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOC bank added
[    0.374199] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOD bank added
[    0.381016] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOE bank added
[    0.387850] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOF bank added
[    0.394625] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOG bank added
[    0.401463] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOH bank added
[    0.408257] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOI bank added
[    0.415098] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOJ bank added
[    0.421889] stm32mp157-pinctrl soc:pin-controller@50002000: GPIOK bank added
[    0.428444] stm32mp157-pinctrl soc:pin-controller@50002000: Pinctrl STM32 initialized
[    0.436604] stm32mp157-pinctrl soc:pin-controller-z@54004000: GPIOZ bank added
[    0.443222] stm32mp157-pinctrl soc:pin-controller-z@54004000: Pinctrl STM32 initialized
```

By default there is no indication in the log that the pin default state has been correctly applied to the device by the base driver. If an issue occurs (like a conflict) the device probe will fail with an error.

- If more kernel logs are needed, use pinctrl dynamic debug:

```
Board $> dmesg -n8
Board $> echo "file drivers/pinctrl* +p" > /sys/kernel/debug/dynamic_debug/control
```

- Since main pin states are applied when devices are probed (meaning before userland prompt) the dynamic printk may need to be enabled in command line:

```
root=/dev/mmcblk0p5 rootwait rw earlyprintk console=ttyS3,115200 loglevel=8 dyndbg="file drivers/pinctrl/* +p"
```

5.3 How to debug

Our pin controller is configured in *strict mode* (meaning that a pin can be requested by only one device). So if a device cannot request a pin during kernel boot, the device tree should be controlled to check if the pin is not affected to two different devices.

Another kind of problem may be that a pin configuration does not fit with the design. In this case, first check the pinconf-pins file in debugfs to verify that the pin hardware settings correspond to the settings defined in the device tree for the same pins. If everything matches, compare the settings with the board schematic in search for missing or unaligned settings, in particular regarding pull-up/pull-down/open-drain ... See GPIO internal peripheral article.

6 Source code location

Source files are located inside kernel Linux.

- **Pinctrl core part**: generic core[3], generic pinconf[4] and generic pinmux[5]
- **STM32 pinctrl vendor part**: folder to STM32 dedicated pinctrl functions[6]
- **base driver part**[2]

7 To go further

7.1 Configure pins for a new board

To configure a new board, two scenarios are possible:

- Pins/groups for device/internal peripherals are already defined: in this case, you only have to select the right group for your device according to schematics.
- Pins/groups for device/internal peripherals are NOT already defined: In this case, you have to define your pins/groups settings inside pincontroller and to select it in your device node according to schematics.
- Please refer to Pinctrl device tree configuration example

Or you can use STM32CubeMX to select your pins and to generate the devicetree accordingly.

7.2 Trainings

More details about pinctrl framework [7]

8 References

- 1.01.1 Documentation/driver-api/pinctl.rst Pinctrl documentation
- 2.02.1 drivers/base/pinctrl.c Pinctrl base driver source
- Pinctrl framework source - core.c Sources of generic pinctrl framework
- Pinctrl framework source - pinconf.c Sources of generic pin configuration
- Pinctrl framework source - pinmux.c Sources of generic pin muxing
- STM32 vendor specific folder Provides all vendor specifics functions
- character device interface, *Linux Kernel and Driver Development* training document, see Introduction to pin muxing **chapter**

Linux® is a registered trademark of Linus Torvalds.

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.)

Debug File System (See https://en.wikipedia.org/wiki/Debugfs for more details)

Application programming interface

foo_driver could be any driver that needs to control a GPIO

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

A quality version of this page, approved on *23 September 2020*, was based off this revision.

1 STM32CubeMX overview

This article describes STM32CubeMX, an official STMicroelectronics graphical software configuration tool.

The STM32CubeMX application helps developers to use the STM32 by means of a user interface, and guides the user through to the initial configuration of a firmware project.

It provides the means to:

- configure pin assignments, the clock tree, or internal peripherals
- simulate the power consumption of the resulting project
- configure and tune DDR parameters
- generate HAL initialization code for Cortex-M4
- generate the Device Tree for a Linux kernel, TF-A and U-Boot firmware for Cortex-A7

It uses a rich library of data from the STM32 microcontroller portfolio.

The application is intended to ease the initial development phase by helping developers to select the best product in terms of features and power.

2 STM32CubeMX main features

- Peripheral and middleware parameters

  Presents options specific to each supported software component

- Peripheral assignment to processors

  Allows assignment of each peripheral to Cortex-A Secure, Cortex-A Non-Secure, or Cortex-M processors

- Power consumption calculator

  Uses a database of typical values to estimate power consumption, DMIPS, and battery life

- Code generation

  Makes code regeneration possible, while keeping user code intact

- Pinout configuration

  Enables peripherals to be chosen for use, and assigns GPIO and alternate functions to pins

- Clock tree initialization

  Chooses the oscillator and sets the PLL and clock dividers

- DDR tunning tool

  Ensures the configuration, testing, and tuning of the MPU DDR parameters. Using U-Boot-SPL Embedded Software.

3 How to get STM32CubeMX

Please, refer to the following link STM32CubeMX to find STM32CubeMX, the Release Note, the User Manual and the product specification.

Doubledata rate (memory domain)

Hardware Abstraction Layer

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Trusted Firmware for Arm® Cortex®-A

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

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.)

Microprocessor Unit

Stable: 19.10.2021 - 13:54 / Revision: 19.10.2021 - 13:54

A quality version of this page, approved on *19 October 2021*, was based off this revision.

Contents

1 Das U-Boot

Das U-Boot ("the Universal Boot Loader" or U-Boot) is an open-source bootloader that can be used on ST boards to initialize the platform and load the Linux® kernel.

- Official website: https://www.denx.de/wiki/U-Boot
- Official manual: U-Boot project documentation and https://www.denx.de/wiki/DULG/Manual
- Official **source code** is available under git repository at [1]

Read the README file before starting using U-Boot. It covers the following topics:

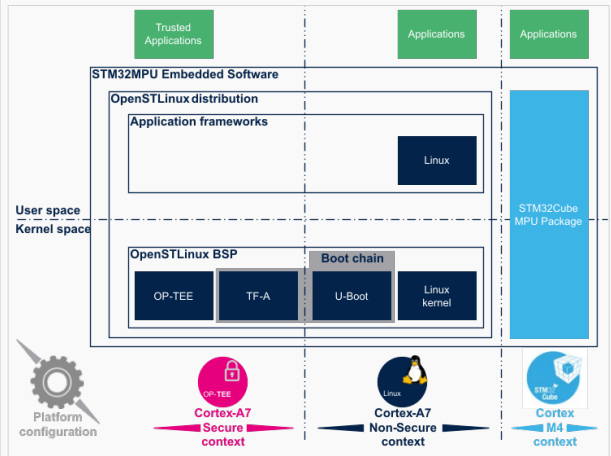
- source file tree structure
- description of CONFIG defines
- instructions for building U-Boot
- brief description of the Hush shell
- list of common environment variables

Do go further, read the documentations available in doc/ and the documentation generated by `make htmldocs` [1].

2 U-Boot overview

The STM32 MPU boot chain uses Trusted Firmware-A (TF-A) as FSBL and U-Boot as SSBL.

The same U-Boot source can also generate an alternate FSBL named SPL. The boot chain becomes: SPL as FSBL and U-Boot as SSBL.



Zoom out to STM32MPU Embedded Software

This alternate boot chain with SPL cannot be used for product development.

2.1 SPL: alternate FSBL

2.1.1 SPL description

The **U-Boot SPL** or **SPL** is an alternate first stage bootloader (FSBL).
It is a small binary (bootstrap utility) generated from the U-Boot source and stored in the internal limited-size embedded RAM.
SPL main features are the following:

- It is loaded by the ROM code.
- It performs the initial CPU and board configuration (clocks and DDR memory).
- It loads the SSBL (U-Boot) into the DDR memory.

2.1.2 SPL restrictions

SPL cannot be used for product development.

SPL is provided only as an example of the simplest FSBL with the objective to support upstream U-Boot development. However, several known limitations have been identified when SPL is used in conjunction with the minimal secure monitor provided within U-Boot for basic boot chain. These limitations apply to:

- power management
- secure access to registers
- limited features (STM32CubeProgrammer / boot from NAND Flash memory)
- SCMI support for clock and reset (not compatible with latest Linux kernel device tree)

There is no workaround for these limitations.

2.1.3 SPL execution sequence

SPL executes the following main steps in SYSRAM:

- **board_init_f()**: driver initialization including DDR initialization (mininimal stack and heap: CONFIG_SPL_STACK_R_MALLOC_SIMPLE_LEN)
- configuration of heap in DDR memory (CONFIG_SPL_SYS_MALLOC_F_LEN)
- **board_init_r()**: initialization of the other drivers activated in the SPL device tree
- loading and execution of U-Boot (or Kernel in Falcon mode[2]: README.falcon).

2.2 U-Boot: SSBL

2.2.1 U-Boot description

U-Boot is the second-stage bootloader (SSBL) of boot chain for STM32 MPU platforms.

SSBL main features are the following:

- It is configurable and expendable.
- It features a simple command line interface (CLI), allowing users to interact over a serial port console.
- It provides scripting capabilities.
- It loads the kernel into RAM and gives control to the kernel.
- It manages several internal and external devices such as NAND and NOR Flash memories, Ethernet and USB.
- It supports the following features and commands:
  - File systems: FAT, UBI/UBIFS, JFFS
  - IP stack: FTP
  - Display: LCD, HDMI, BMP for splashcreen
  - USB: host (mass storage) or device (DFU stack)

2.2.2 U-Boot execution sequence

U-Boot executes the following main steps in DDR memory:

- **Pre-relocation** initialization (common/board_f.c): minimal initialization (such as CPU, clock, reset, DDR and console) running at the CONFIG_SYS_TEXT_BASE load address.
- **Relocation**: copy of the code to the end of DDR memory.
- **Post-relocation initialization**:(common/board_r.c): initialization of all the drivers.
- **Command execution** through autoboot (CONFIG_AUTOBOOT) or console shell.
  - Execution of the boot command (by default bootcmd=CONFIG_BOOTCOMMAND):

  for example, execution of the command `bootm` to:
    - load and check images (such as kernel, device tree and ramdisk)
    - fixup the kernel device tree
    - install the secure monitor (optional) or
    - pass the control to the Linux kernel (or to another target application)

3 U-Boot configuration

The U-Boot binary configuration is based on

- **Kbuild infrastructure** (as in Linux Kernel, you can use `make menuconfig` in U-Boot)

The configurations are based on:

  - options defined in Kconfig files (CONFIG_ compilation flags)
  - the selected configuration file: configs/stm32mp*_defconfig

- **other compilation flags** defined in include/configs/stm32mp*.h (these flags are progressively migrated to Kconfig)

The file name is configured through CONFIG_SYS_CONFIG_NAME.

For STM32MP15x lines, the include/configs/stm32mp1.h file is used.

- DeviceTree: U-Boot binaries include a device tree blob that is parsed at runtime

All the configuration flags (prefixed by CONFIG_) are described in the source code, either in the README file or in the documentation directory .

For example, CONFIG_SPL activates the SPL compilation.

Hence to compile U-Boot, select the <target> and the device tree for the board in order to choose a predefined configuration.

Refer to #U-Boot_build for examples.

3.1 Kbuild

Like the kernel, the U-Boot build system is based on configuration symbols (defined in Kconfig files). The selected values are stored in a **.config** file located in the build directory, with the same makefile target. .

Proceed as follows:

- Select a predefined configuration (defconfig file in configs directory) and generate the first **.config**:

```
PC $> make <config>_defconfig.
```

- Change the U-Boot compile configuration (modify .config) by using one of the following five `make` commands:

```
PC $> make menuconfig --> menu based program
PC $> make config  --> line-oriented configuration
PC $> make xconfig --> QT program[3]
PC $> make gconfig --> GTK program
PC $> make nconfig --> ncurse menu based program
```

You can then compile U-Boot with the updated .config.

Warning: the modification is performed locally in the build directory. It will be lost after a `make distclean`.

Save your configuration to be able to use it as a defconfig file:

```
PC $> make savedefconfig
```

This target saves the current config as a defconfig file in the build directory. It can then be compared with the predefined configuration (configs/stm32mp*defconfig).

The other makefile targets are the following:

```
 PC $> make help
....
Configuration targets:
  config          - Update current config utilising a line-oriented program
  nconfig         - Update current config utilising a ncurses menu based
                    program
  menuconfig          - Update current config utilising a menu based program
  xconfig          - Update current config utilising a Qt based front-end
  gconfig          - Update current config utilising a GTK+ based front-end
  oldconfig          - Update current config utilising a provided .config as base
  localmodconfig  - Update current config disabling modules not loaded
  localyesconfig  - Update current config converting local mods to core
  defconfig          - New config with default from ARCH supplied defconfig
  savedefconfig   - Save current config as ./defconfig (minimal config)
  allnoconfig           - New config where all options are answered with no
  allyesconfig           - New config where all options are accepted with yes
  allmodconfig           - New config selecting modules when possible
  alldefconfig    - New config with all symbols set to default
  randconfig          - New config with random answer to all options
  listnewconfig   - List new options
  olddefconfig           - Same as oldconfig but sets new symbols to their
                   default value without prompting
```

3.2 Device tree

Refer to doc/README.fdt-control for details.

The board device tree has the same binding as the kernel. It is integrated within the U-Boot binaries: u-boot.bin

- By default, it is appended at the end of the code (CONFIG_OF_SEPARATE).
- It can be embedded in the U-Boot binary (CONFIG_OF_EMBED). This is particularly useful for debugging since it enables easy .elf file loading.

The U-Boot device tree (u-boot.dtb) can be also provided as external file loaded by FSBL when U-Boot code is started (u-boot-nodtb.bin: code without device tree): device tree address is provided as boot parameter (in r2 register).

A default device tree is available in the defconfig file (by setting CONFIG_DEFAULT_DEVICE_TREE).

You can either select another supported device tree using the DEVICE_TREE make flag. For stm32mp boards, the corresponding file is <dts-file-name>.dts in arch/arm/dts/stm32mp*.dts , with <dts-file-name> set to the full name of the board:

```
PC $> make DEVICE_TREE=<dts-file-name>
```

or provide a device tree blob (dtb file) resulting from the dts file compilation, by using the EXT_DTB option:

```
PC $> make EXT_DTB=boot/<dts-file-name>.dtb
```

The SPL device tree is also generated from this device tree. However to reduce its size, the U-Boot makefile uses the fdtgrep tool to parse the full U-Boot DTB and identify all the drivers required by SPL.

To do this, U-Boot uses specific device-tree flags to determine if the associated driver is initialized prior to U-Boot relocation and /or if the associated node is present in SPL :

- **u-boot,dm-pre-reloc** => present in SPL, initialized before relocation in U-Boot
- **u-boot,dm-pre-proper** => initialized before relocation in U-Boot
- **u-boot,dm-spl** => present in SPL

In the device tree used by U-Boot, these flags **need to be added in all the nodes** used in SPL or in U-Boot before relocation, and for all used handles (clock, reset, pincontrol).

To obtain a device tree file **<dts-file-name>.dts** that is identical to the Linux kernel one, these U-Boot properties are only added for ST boards in the add-on file **<dts-file-name>-u-boot.dtsi**. This file is automatically included in **<dts-file-name>.dts** during device tree compilation (this is a generic U-Boot Makefile behavior).

4 U-Boot command line interface (CLI)

Refer to U-Boot Command Line Interface.

If CONFIG_AUTOBOOT is activated, you have CONFIG_BOOTDELAY seconds (2s by default, 1s for ST configuration) to enter the console by pressing any key, after the line below is displayed and bootcmd is executed (CONFIG_BOOTCOMMAND):

```
Hit any key to stop autoboot:  2
```

4.1 Commands

The commands are defined in cmd/*.c . They are activated through the corresponding **CONFIG_CMD_*** configuration flag.

Use the `help` command in the U-Boot shell to list the commands available on your device:

```
Board $> help
```

Below the list of all commands extracted from U-Boot Manual (**not-exhaustive**):

- Information Commands
  - bdinfo - prints Board Info structure
  - coninfo - prints console devices and information
  - flinfo - prints Flash memory information
  - iminfo - prints header information for application image
  - help - prints online help
- Memory Commands
  - base - prints or sets the address offset
  - crc32 - checksum calculation
  - cmp - memory compare
  - cp - memory copy
  - md - memory display
  - mm - memory modify (auto-incrementing)
  - mtest - simple RAM test
  - mw - memory write (fill)
  - nm - memory modify (constant address)
  - loop - infinite loop on address range
- Flash Memory Commands
  - cp - memory copy
  - flinfo - prints Flash memory information
  - erase - erases Flash memory
  - protect - enables or disables Flash memory write protection
  - mtdparts - defines a Linux compatible MTD partition scheme
- Execution Control Commands
  - source - runs a script from memory
  - bootm - boots application image from memory

- go - starts application at address 'addr'

- Download Commands
  - bootp - boots image via network using BOOTP/TFTP protocol
  - dhcp - invokes DHCP client to obtain IP/boot params
  - loadb - loads binary file over serial line (kermit mode)
  - loads - loads S-Record file over serial line
  - rarpboot- boots image via network using RARP/TFTP protocol
  - tftpboot- boots image via network using TFTP protocol
- Environment Variables Commands
  - printenv- prints environment variables
  - saveenv - saves environment variables to persistent storage
  - setenv - sets environment variables
  - run - runs commands in an environment variable
  - bootd - default boot, that is run 'bootcmd'
- Flattened Device Tree support
  - fdt addr - selects the FDT to work on
  - fdt list - prints one level
  - fdt print - recursive printing
  - fdt mknode - creates new nodes
  - fdt set - sets node properties
  - fdt rm - removes nodes or properties
  - fdt move - moves FDT blob to new address
  - fdt chosen - fixup dynamic information
- Special Commands
  - i2c - I2C sub-system
- Storage devices
- Miscellaneous Commands
  - echo - echoes args to console
  - reset - performs a CPU reset
  - sleep - delays the execution for a predefined time
  - version - prints the monitor version

To add a new command, refer to doc/README.commands .

4.2 U-Boot environment variables

The U-Boot behavior is configured through environment variables.

Refer to Manual and README / Environment Variables.

On the first boot, U-Boot uses a default environment embedded in the U-Boot binary. You can modify it by changing the content of CONFIG_EXTRA_ENV_SETTINGS in your configuration file (for example ./include/configs/stm32mp1.h) (see README / - Default Environment).

This environment can be modified and saved in the boot device. When it is present, it is loaded during U-Boot initialization:

- To boot from *e*•MMC/SD card (CONFIG_ENV_IS_IN_MMC): at the end of the partition indicated by config field "u-boot,mmc-env-partition" in device-tree (for ST boards: partition named "fip" in ecosystem release v3.0.0 with FIP support or partition named "ssbl" without FIP support).

- To boot from NAND Flash memory (CONFIG_ENV_IS_IN_UBI): in the two UBI volumes "config" (CONFIG_ENV_UBI_VOLUME) and "config_r" (CONFIG_ENV_UBI_VOLUME_REDUND).
- To boot from NOR Flash memory (CONFIG_ENV_IS_IN_SPI_FLASH): the u-boot_env mtd parttion (at offset CONFIG_ENV_OFFSET).

4.2.1 env command

The `env` command allows displaying, modifying and saving the environment in U-Boot console.

```
Board $> help env
env - environment handling commands

Usage:
env default [-f] -a - [forcibly] reset default environment
env default [-f] var [...] - [forcibly] reset variable(s) to their default values
env delete [-f] var [...] - [forcibly] delete variable(s)
env edit name - edit environment variable
env exists name - tests for existence of variable
env print [-a | name ...] - print environment
env print -e [name ...] - print UEFI environment
env run var [...] - run commands in an environment variable
env save - save environment
env set -e name [arg ...] - set UEFI variable; unset if 'arg' not specified
env set [-f] name [arg ...]
```

Example: proceed as follows to restore the default environment and save it. This is useful after a U-Boot upgrade:

```
Board $> env default -a
Board $> env save
```

You can also use the command activated by CONFIG_CMD_ERASEENV:

```
Board $> env erase
```

4.2.2 bootcmd

"bootcmd" variable is the autoboot command. It defines the command executed when U-Boot starts (CONFIG_BOOTCOMMAND).

For stm32mp, CONFIG_BOOTCOMMAND="run bootcmd_stm32mp":

```
Board $> env print bootcmd
 bootcmd=run bootcmd_stm32mp
```

"bootcmd_stm32mp" is a script that selects the command to be executed for each boot device (see ./include/configs/stm32mp1.h), based on generic distro scripts:

- To boot from a serial/usb device: execute the `stm32prog` command.
- To boot from an *e*•MMC, SD card: boot only on the same device (bootcmd_mmc...).
- To boot from a NAND Flash memory: boot on ubifs partition on the NAND memory (bootcmd_ubi0).
- To boot from a NOR Flash memory: use the SD card (on SDMMC 0 on ST boards with bootcmd_mmc0)

```
Board $> env print bootcmd_stm32mp
```

You can then change this configuration:

- either permanently in your board file
  - default environment by CONFIG_EXTRA_ENV_SETTINGS (see ./include/configs/stm32mp1.h)
  - change CONFIG_BOOTCOMMAND value in your defconfig

```
CONFIG_BOOTCOMMAND="run bootcmd_mmc0"
```

```
CONFIG_BOOTCOMMAND="run distro_bootcmd"
```

- or temporarily in the saved environment:

```
Board $> env set bootcmd run bootcmd_mmc0
Board $> env save
```

Note: To reset the environment to its default value:

```
Board $> env default bootcmd
Board $> env save
```

4.3 Generic Distro configuration

Refer to doc/README.distro for details.

This feature is activated by default on ST boards (CONFIG_DISTRO_DEFAULTS):

- one boot command (bootmcd_xxx) exists for each bootable device.
- U-Boot is independent from the Linux distribution used.
- bootcmd is defined in ./include/config_distro_bootcmd.h

When DISTRO is enabled, the command that is executed by default is include/config_distro_bootcmd.h :

```
bootcmd=run distro_bootcmd
```

This script tries any device found in the 'boot_targets' variable and executes the associated bootcmd.

Example for mmc0, mmc1, mmc2, pxe and ubifs devices:

```
bootcmd_mmc0=setenv devnum 0; run mmc_boot
bootcmd_mmc1=setenv devnum 1; run mmc_boot
bootcmd_mmc2=setenv devnum 2; run mmc_boot
bootcmd_pxe=run boot_net_usb_start; dhcp; if pxe get; then pxe boot; fi
bootcmd_ubifs0=setenv devnum 0; run ubifs_boot
```

U-Boot searches for an **extlinux.conf** configuration file for each bootable device. This file defines the kernel configuration to be used with `bootm` command:

- bootargs
- files to start the OS:
  - kernel (uImage) + device tree + ramdisk files (optional)

- FIT image, including all these needed files (for details see doc/uImage.FIT/howto.tx)

4.4 U-Boot scripting capabilities

"Script files" are command sequences that are executed by the U-Boot command interpreter. This feature is particularly useful to configure U-Boot to use a real shell (hush) as command interpreter.

See U-Boot script manual for an example.

5 U-Boot build

See U-Boot Documentation.

5.1 Prerequisites

- a PC with Linux and tools:
  - see PC_prerequisites
  - #ARM cross compiler
- U-Boot source code
  - the latest STMicroelectronics U-Boot version
    - tar.xz file from Developer Package (for example STM32MP1) or from latest release on ST github [4]
    - from GITHUB[5], with `git` command

```
PC $> git clone https://github.com/STMicroelectronics/u-boot
```

  - from the Mainline U-Boot in official GIT repository [6]

```
PC $> git clone https://source.denx.de/u-boot/u-boot.git
```

5.2 ARM cross compiler

A cross compiler [7] must be installed on your Host (X86_64, i686, ...) for the ARM targeted Device architecture. In addition, the $PATH and $CROSS_COMPILE environment variables must be configured in your shell.

You can use gcc for ARM, available in:

- the SDK toolchain (see Cross-compile with OpenSTLinux SDK)

PATH and CROSS_COMPILE are automatically updated.

- an existing package

For example, install gcc-arm-linux-gnueabihf on Ubuntu/Debian: (**PC $>** sudo apt-get.

- an existing toolchain:
  - latest gcc toolchain provided by arm (https://developer.arm.com/open-source/gnu-toolchain/gnu-a/downloads/)
  - gcc v7 toolchain provided by linaro: (https://www.linaro.org/downloads/)

For example, to use *gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf.tar.xz* from arm, extract the toolchain in $HOME and update your environment with:

```
PC $> export PATH=$HOME/gcc-arm-9.2-2019.12-x86_64-arm-none-linux-gnueabihf/bin:$PATH
PC $> export CROSS_COMPILE=arm-none-linux-gnueabihf-
```

For example, to use **gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi.tar.xz**
from https://releases.linaro.org/components/toolchain/binaries/7.2-2017.11/arm-linux-gnueabi/
Unzip the toolchain in $HOME and update your environment with:

```
PC $> export PATH=$HOME/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/bin:$PATH
PC $> export CROSS_COMPILE=arm-linux-gnueabi-
```

5.3 Compilation

In the U-Boot source directory, select the defconfig for the <target> and the <device tree> for your board and then execute the `make all` command:

```
PC $> make <target>_defconfig
PC $> make DEVICE_TREE=<device tree> all
```

Use help to list other targets than all:

```
PC $> make help
```

Optionally

- **KBUILD_OUTPUT** can be used to change the output build directory in order to compile several targets in the source directory. For example:

```
PC $> export KBUILD_OUTPUT=<path>
```

- **DEVICE_TREE** can also be exported to your environment when only one board is supported. For example:

```
PC $> export DEVICE_TREE=<device-tree>
```

The result is the following:

```
PC $> export KBUILD_OUTPUT=<path>
PC $> export DEVICE_TREE=<device tree>
PC $> make <target>_defconfig
PC $> make all
```

Examples from STM32MP15 U-Boot:

The boot chain for STM32MP15x lines use stm32mp15_trusted_defconfig:

```
PC $> make stm32mp15_trusted_defconfig
PC $> make DEVICE_TREE=stm32mp157f-dk2 all
```

```
PC $> export KBUILD_OUTPUT=../build/stm32mp15_trusted
PC $> export DEVICE_TREE=stm32mp157c-ev1
PC $> make stm32mp15_trusted_defconfig
PC $> make all
```

5.4 Output files

The resulting U-Boot files are located in your build directory (U-Boot or KBUILD_OUTPUT).

Since ecosystem release v3.0.0 , two U-Boot files are used by ST boards to generate FIP used by FSBL TF-A, with or without OP-TEE support:

- BL33_CFG=u-boot.dtb : the U-Boot device tree, selected by DEVICE_TREE, loaded by TF-A BL2 and amended by secure monitor (SPMIN or OP-TEE)
- BL33=u-boot-nodtb.bin: the U-Boot executable, loaded by TF-A BL2 started by secure monitor with BL33_CFG as parameter

Nota: All the compiled device tree are available in $KBUILD_OUTPUT/arch/arm/dts/*.dtb.
You can select them as BL33_CFG without U-Boot recompilation.

See TF-A_overview for FIP details.

The file used to debug with gdb is

- u-boot : elf file for U-Boot

For ecosystem release v2.1.0 : u-boot.stm32 : U-Boot binary with STM32 image header, including device tree selected by DEVICE_TREE, loaded by TF-A

This behavior can be restored if you activate CONFIG_STM32MP15x_STM32IMAGE in your defconfig of ecosystem release v3.0.0

This temporary option is only introduced to facilitate the FIP migration but it will be removed in the next EcosystemRelease.

The STM32 image format (*.stm32) is managed by mkimage U-Boot tools and Signing_tool. It is requested by ROM code and TF-A without FIP support (see STM32 header for binary files for details).

6 References

- https://u-boot.readthedocs.io/en/stable/index.html
- https://www.denx.de/wiki/pub/U-Boot/MiniSummitELCE2013/2013-ELCE-U-Boot-Falcon-Boot.pdf
- https://en.wikipedia.org/wiki/Xconfig
- https://github.com/STMicroelectronics/u-boot/releases
- https://github.com/STMicroelectronics/u-boot
- https://source.denx.de/u-boot/u-boot.git or https://github.com/u-boot/u-boot
- https://en.wikipedia.org/wiki/Cross_compiler

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Linux® is a registered trademark of Linus Torvalds.

First Stage Boot Loader

Secondary Program Loader, *Also known as* ***U-Boot SPL***

Second Stage Boot Loader

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Read Only Memory

Central processing unit

Doubledata rate (memory domain)

Flash memories combine high density and cost effectiveness of EPROMs with the electrical erasability of EEPROMs. For this reason, the Flash memory market is one of the most exciting areas of the semiconductor industry today and new applications requiring in system reprogramming, such as cellular telephones, automotive engine management systems, hard disk drives, PC BIOS software for Plug & Play, digital TV, set top boxes, fax and other modems, PC cards and multimedia CD-ROMs, offer the prospect of very high volume demand.

System control and management interface

Microprocessor Unit

High-Definition Multimedia Interface (HDMI standard)

Device Firmware Upgrade

Device Tree Binary (or Blob)

Memory Technology Device

Trivial File Transfer Protocol (https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol)

Dynamic Host Configuration Protocol (See https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol for more details)

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

MultimediaCard

SD memory card (https://www.sdcard.org)

Firmware Image Package is a packaging format used by TF-A

Serial Peripheral Interface

Operating System

Flattened uImage Tree is a packaging format used by U-Boot

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Trusted Firmware for Arm® Cortex®-A

Open Portable Trusted Execution Environment

Boot Loader stage 3-3

Boot Loader stage 2