NuMicro® Family

Arm® Cortex®-A35-based Microcontroller

NuMicro® Family
MA35D1 U-Boot
User Manual

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 OVERVIEW

The U-Boot utility is a multi-platform, open-source, universal boot-loader with comprehensive support for loading and managing boot images, such as the Linux kernel. It supports the following features including:

- Network download
- Serial download
- Flash access
- Memory utility
- Interactive shell

This document introduces the U-Boot features that are related with MA35D1. Chapter 2 and 3 cover the U-Boot configuration and device tree modification. Chapter 4 lists the commands for building and cleaning U-Boot, and Chapter 5 shows the command sequence of bringing Linux kernel from different storage devices. For more information about U-Boot, please visit its official website: https://www.denx.de/wiki/U-Boot.

2 U-BOOT CONFIGURATION

There is a default configuration for the MA35D1 series chips provided by Nuvoton. Before modifying any configuration of U-Boot, it is recommended to load the default configuration of U-Boot first. User can type “make ma35d1_defconfig” command to do that. Sometimes if system cannot boot up, user can load the default configuration to recovery U-Boot to safe status.

```
$ make ma35d1_defconfig
```

Sometimes fine-tune U-Boot configuration, for example to enable some features that are not enabled by default, or to remove unused feature to reduce image size. U-Boot provides an interface to enter configuration menu by typing “make menuconfig” command.

```
$ make menuconfig
```

This is a multi-layer menu in configuration system. In the current page, user can press arrow keys to control the layer of configuration system. Select U-Boot function by pressing “up” or “down” key and select menu function in the bottom of page by pressing “left” or “right” key. To enter the next layer of configuration page, user can press “enter” key.

There are five functions at the bottom of menu page. User can disable or enable U-Boot function by pressing space key when cursor stays at “Select”. The symbol in front of the selection function “[ ]” stands for this function is disabled, “[*]” stands for this function is enabled.

The menu page can be returned to upper layer by pressing space key when cursor stays at “Exit” at the bottom of menu page. If it’s at the top layer of configuration system, system will inform user if wants to save the configuration and exit.

The help screen will show when cursor is at “Help” by pressing space key. To save current configuration or load old configuration, use can press space key when cursor is at “Save” or “Load” at the bottom of menu page.

The U-Boot configuration file will be named “.config” and be saved in the U-Boot source tree directory.

3 U-BOOT OPTIONS

This chapter describes the menuconfig options to enable MA35D1 drivers and their device tree setting.

3.1 System

3.1.1 Memory

The DDR memory size accessible by U-Boot is defined in device tree. This size is not equal to the total DDR size of the system because some part of the DDR is reserved for secure access only so U-Boot executing at EL2 cannot access that region. The secure DDR region reserved by TF-A is configurable and is 32MB by default. So the system memory size defined in device tree should be total DDR size – 32MB. Below is the memory setting of a system with 256MB DDR. The starting address attribute is 0x80000000 which is the DDR starting address in system map, and the size attribute is 256MB – 32MB = 224MB = 0xe000000.

```
	memory {
		device_type = "memory";
		reg = <0x00000000 0x80000000 0 0x0e000000>;
	};
```

3.1.2 System controller

A system control node should be added in the device tree allowing other driver to get the base address of system control registers. Here shows the system control node of MA35D1 in device tree.

```
	sys: system-management@40460000 {
		compatible = "nuvoton,ma35d1-sys", "syscon", "simple-mfd";
		reg = <0x0 0x40460000 0x0 0x200>;
	}
```

3.1.3 Clock controller

A driver source code in U-Boot is in charge of control the engine clock of MA35D1 peripheral clocks. The source code:

- drivers/clk/nuvovon/clk_ma35d1.c

This clock driver operates under Common Clock Framework (CCF) subsystem. The menuconfig options to enable CCF and MA35D1 clock driver are listed below.

```
Device Drivers  --->
  Clock  --->
    [*] Enable clock driver support
    -*- Common Clock Framework [CCF] support
    [*] MA35D1 Common Clock Framework [CCF] support
```

There are several device tree nodes represent the clock source and clock controller in MA35D1. Each root clock source, HXT, LXT, HIRC, and LIRC has its own node using “clock-frequency” to record its clock frequency and “clock-output-names” denotes its clock gating register bit field.

```
	clk_hxt:hxt {
		compatible = "fixed-clock";
		#clock-cells = <0>;
		clock-frequency = <24000000>;
```

```
            clock-output-names = "hxt_gate";
        };

        clk_lxt:lxt {
            compatible = "fixed-clock";
            #clock-cells = <0>;
            clock-frequency = <32768>;
            clock-output-names = "lxt_gate";
        };

        clk_hirc:hirc {
            compatible = "fixed-clock";
            #clock-cells = <0>;
            clock-frequency = <12000000>;
            clock-output-names = "hirc_gate";
        };

        clk_lirc:lirc {
            compatible = "fixed-clock";
            #clock-cells = <0>;
            clock-frequency = <32000>;
            clock-output-names = "lirc_gate";
        };
```

And there is a node that describes the attribute of clock controller.

```
        clk: clock-controller@40460200 {
```

“compatible” must be set to “nuvoton,ma35d1-clk”, “syscon”, and “simple-mfd”.

```
            compatible = "nuvoton,ma35d1-clk", "syscon", "simple-mfd";
```

“reg” defines the base address and size of clock control register. The base address and size is 0x40460200 and 0x100 respectively.

```
            reg = <0x00000000 0x40460200 0x0 0x100>;
```

Following attributes are used to set the clock gate of root clock sources.

```
            #clock-cells = <1>;
            clocks = <&clk_hxt>, <&clk_lxt>, <&clk_hirc>,
                     <&clk_lirc>;
            clock-names = "hxt_gate", "lxt_gate", "hirc_gate",
                          "lirc_gate";
```

“nuvoton.sys” is used to assign the system controller base address.

```
            nuvoton,sys = <&sys>;
        };
```

3.2 Timers

3.2.1 Watchdog Timer

The following lists the driver source codes of MA35D1 Watchdog Timer:

- drivers/watchdog/ma35d1_wdt.c

To enable WDT support in U-Boot, please enable the following options in configuration interface.

```
Device Drivers  --->
  Watchdog Timer support  --->
    [*] Enable U-Boot watchdog reset
    (60000) Watchdog timeout in msec
    [*] Enable driver model for watchdog timer drivers
    [*] MA35D1 watchdog timer support
```

To use WDT interface in U-Boot, a node that describes the attribute of WDT needs to be present. Here describes the meaning of each attribute of WDT.

The WDT can work in non-secure mode is WDT1, so here shows the base address of WDT1 40440000.

```
      wdt1: wdt@40440000 {
```

“compatible” must be set to “nuvoton,ma35d1-wdt”.

```
            compatible = "nuvoton,ma35d1-wdt";
```

“reg” defines the base address and size of WDT1 register map. The base address is 0x40440000 for WDT1.

```
            reg = <0x0 0x40440000 0x0 0x100>;
```

This attribute is to allow WDT driver to get the system control register base address.

```
            nuvoton,ma35d1-sys = <&sys>;
```

The following attribute is for driver to enable the WDT1 clock setting and must not change.

```
            clocks = <&clk wdt1_gate>;
            clock-names = "wdt";
```

Set “okay” to enable WDT, “disabled” to disable WDT.

```
            status = "okay";
      };
```

3.2.2 RTC

U-Boot supports using “rtc” commands to set and get data/time. The driver supports access MA35D1 RTC is

- drivers/rtc/ma35d1_rtc.c

Please enable the following menuconfig option to include RTC driver in U-Boot binary.

```
Device Drivers  --->
  Real Time Clock  --->
    [*] Enable Driver Model for RTC drivers
    [*] Enable MA35D1 RTC driver
```

In order to let U-Boot enable RTC driver after booting, a RTC node in device tree must be set to “okay”

```
        rtc: rtc@40410000 {
```

“compatible” must be set to “nuvoton,ma35d1-rtc”.

```
                compatible = "nuvoton,ma35d1-rtc";
```

“reg” defines the base address and size of RTC register map. The base address is 0x40410000 for RTC.

```
                reg = <0x0 0x40410000 0x0 0x10000>;
                status = "okay";
        };
```

3.3 I/O

3.3.1 Pinctrl and GPIO

Pinctrl and GPIO driver are used to configure the multi-function pin settings and control the GPIO pin attributes. The driver source codes are:

- drivers/pinctrl/nuvoton/pinctrl-nvt.c
- drivers/pinctrl/nuvoton/pinctrl-ma35d1.c
- drivers/gpio/ma35d1-gpio.c

To enable pinctrl and GPIO support in U-Boot, please enable following options in configuration interface.

```
Device Drivers  --->
  Pin controllers  --->
    [*] Support pin controllers
    [*]   Support full pin controllers
    [*]     Support generic pin controllers
    [*]       Support pin multiplexing controllers
    [*]       Support pin configuration controllers
    [*]     Support recursive binding for pin configuration nodes
    [*] NUVOTON MA35D1 pin control drivers
  GPIO Support  --->
    [*] Enable Driver Model for GPIO drivers
    [*] MA35D1 GPIO driver
```

Below is a reference of the pinctrl and GPIO node defined in device tree for MA35D1.

```
        pinctrl: pinctrl {
```

“compatible” must be set to “nuvoton,ma35d1-pinctrl”.

```
                compatible = "nuvoton,ma35d1-pinctrl";
```

This attribute is to allow pinctrl driver to get the system control registers base and access multi-function pin control registers.

```
                nuvoton,sys = <&sys>;
```

“#address-cells” and “#size-cells” should be set to 2.

```
                #address-cells = <2>;
```

```
            #size-cells = <2>;
            status = "okay";
```

The MA35D1 has 14 GPIO ports total, from A ~ N. Each port has to declare its register base and clock port in device tree. Below is the declaration example for port A. Where its register base address is 0x40040000, register map size 0x40, and clock fate is “gpa_gate”.

```
            gpioa: gpioa@40040000 {
                  reg = <0x0 0x40040000 0 0x40>,
                        <0x0 0x40040800 0 0x40>;
                  clocks = <&clk gpa_gate>;
                  gpio-controller;
                  #gpio-cells = <2>;
            };
            ...
```

“pcfg_default” defines a pin configuration with default high rate, Schmitt trigger disabled and pull up/down disabled.

```
            pcfg_default: pcfg-default {
```

Set “slew-rate” to 0 for normal slew rate and 1 to high slew rate.

```
                  slew-rate = <1>;
```

Add “input-schmitt-disable” to disable Schmitt trigger and “input-schmitt-enable” to enable Schmitt trigger.

```
                  input-schmitt-disable;
```

Add “bias-disable” to disable internal pull up and pull down. Add “bias-pull-up” to enable internal pull up, add “bias-pull-down” to enable internal pull down.

```
                  bias-disable;
```

“power-source” could be either 3300 or 1800 depending on the I/O voltage is 3.3V or 1.8V.

```
                  power-source = <3300>;
```

“drive-strength” controls the output driving strength, valid range is between 0~7, the higher the stringer. And 3 is the default setting.

```
                  drive-strength = <3>;
            };
```

Here is an example for GMAC pins configuration.

```
            pcfg_emac_1_8V: pcfg-pcfg_emac_1_8V {
                  slew-rate = <0>;
                  input-schmitt-enable;
                  bias-disable;
                  power-source = <1800>;
                  drive-strength = <1>;
            }
            ...
```

All peripheral using external pins has to add a node to declare the pins it is using and the multi-function

pin setting. In following example, UART0 register 2 pins, PE14 and PE15 for TX and RX function respectively with default pin configuration “pcfg_default”. All MA35D1 multi-function pin definitions are defined in dt-bindings/pinctrl/ma35d1-pinfunc.h, which is included at the begin of MA35D1 device tree file.

```
            uart0 {
                  pinctrl_uart0: uart0grp {
                        nuvoton,pins =
                        <SYS_GPE_MFPH_PE14MFP_UART0_TXD &pcfg_default>,
                        <SYS_GPE_MFPH_PE15MFP_UART0_RXD &pcfg_default>;
                  };
            };
      };
```

3.4 Connectivity

3.4.1 UART

UART is used as a console to show system information and the fundamental communication interface for CLI, and the MA35D1 only provides port 0 for the console. The driver of MA35D1 UART is:

- drivers/serial/serial_nuvoton.c

The following lists the U-Boot options to enable UART0 as console.

```
Device Drivers  --->
  Serial drivers  --->
    (115200) Default baudrate
    [*] Require a serial port for console
    [*] Specify the port number used for console
    [*] Provide a serial driver
    (0) UART used for console
    [*] Enable Driver Model for serial drivers
    [*] Nuvoton UART support
```

The node that describes the attribute of UART0 in device is shown here with description of its elements.

The base address of UART0 is set to 40700000.

```
      uart0: serial@40700000 {
```

Following attribute is to set the MFP for UART0 interface. The pins used by UART0 is defined in pinctrl node.

```
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_uart0>;
```

“compatible” must be set to “nuvoton,ma35d1-uart”.

```
            compatible = "nuvoton,ma35d1-uart";
```

“reg” defines the base address and size of UART0 control register.

```
            reg = <0x0 0x40700000 0x0 0x10000>;
```

Set “okay” to enable UART interface, “disable” to disable UART controller.

```
            status = "okay";
        };
```

3.4.2 QSPI

The driver of MA35D1 QSPI is:

- drivers/spi/ma35d1_spi.c

To enable SPI support in U-Boot, please enable following options in configuration interface.

```
Device Drivers  --->
  [*] SPI Support  --->
    [*]   Enable Driver Model for SPI drivers
    -*-   SPI memory extension
    [*]   MA35D1 SPI driver
```

The following lists the U-Boot options to enable SPI NOR Flash.

```
Device Drivers  --->
  MTD Support  --->
    [*] Enable MTD layer
    [*] Enable Driver Model for MTD drivers
        SPI Flash Support  --->
            [*] Enable Driver Model for SPI flash
            [*] SPI Flash Core Interface support
            [*]   Atmel SPI flash support
            [*]   EON SPI flash support
            [*]   GigaDevice SPI flash support
            [*]   ISSI SPI flash support
            [*]   Macronix SPI flash support
            [*]   Spansion SPI flash support
            [*]   STMicro SPI flash support
            [*]   SST SPI flash support
            [*]   Winbond SPI flash support
            [*]   XMC SPI flash support
            [*]   SPI Flash MTD support
```

The following lists the U-Boot options to enable SPI NAND Flash.

```
Device Drivers  --->
  MTD Support  --->
    [*] Enable MTD layer
    [*] Enable Driver Model for MTD drivers
        [*] SPI NAND device Support  ----
```

Below is the example of device tree node for QSPI.

```
      qspi0: qspi@40680000 {
```

“compatible” should be set to "nuvoton,ma35d1-qspi" for QSPI ports.

```
            compatible = "nuvoton,ma35d1-qspi";
```

Register base address for QSPI0 is 0x40680000.

```
            reg = <0x0 0x40680000 0x0 0x10000>;
            reg-names = "qspi_base";
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_qspi0>;
```

These entries assign the QSPI clock gate.

```
            nuvoton,clk = <&clk>;
            clocks = <&clk qspi0_gate>;
            clock-names = "qspi0";
```

The following settings should keep as is.

```
            u-boot,dm-pre-reloc;
            status = "okay";

            #address-cells = <1>;
            #size-cells = <0>;
```

Below is an SPI NAND Flash example. The setting should modify these elements according to the Flash mounted on the hardware.

```
            spi-nand@0 {
                  #address-cells = <1>;
                  #size-cells = <1>;
                  compatible = "spi-nand";
                  spi-max-frequency = <30000000>;
                  spi-tx-bus-width = <1>;
                  spi-rx-bus-width = <4>;
                  reg = <0>;
            };
      };
```

Although MA35D1 access SPI NAND flash through SPI interface, it's storage device is still NAND flash and may contain bad blocks. So user should use U-Boot's "mtd" commands to access SPI NAND flash instead of "sf" commands. Because "sf" commands do not support bad block handling.

3.4.3 I²C

The driver source code controls the MA35D1 I²C interface in U-Boot is:

- drivers/i2c/ma35d1_i2c.c

To enable I²C support in U-Boot, please enable the following options in configuration interface.

```
Device Drivers  --->
  I2C Support  --->
    [*] Enable Driver Model for I2C drivers
```

```
    [*] MA35D1 I2C driver
```

Here is an example of device tree node that describes the feature of MA35D1 I²C interface 1.

```
      i2c1: i2c@40810000 {
```

“compatible” must be set to “nuvoton,ma35d1-i2c”.

```
            compatible = "nuvoton,ma35d1-i2c";
```

“reg” defines the base address and size of I²C control register. The base address of interface X is 0x40800000 + 0x10000 * X, where X = 0~5. So the base address of interface 1 is 0x40810000.

```
            reg = <0x0 0x40810000 0x0 0x10000>;
```

The following two attributes describe the MFP for I²C interface. The pins used by I²C is defined in pinctrl node.

```
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_i2c0>;
```

This attribute defined the peripheral clock gating bit.

```
            clocks = <&clk i2c0_gate>;
```

“clock-frequency” defines the I²C bus clock. The clock rate defined here should not exceed the maximum clock that I2C slave supports. In this example, the clock frequency is for a slave device that supports standard mode. Set status to “okay” to enable I²C controller, “disable” to disable I²C controller.

```
            clock-frequency = <100000>;
            status = "okay";
      };
```

3.5 Networking

U-Boot supports fundamental network features such as DHCP to get a IP address lease, TFTP enable user to down image over the Ethernet. PING to check network connection status, and NFS to access file over Ethernet.

3.5.1 GMAC

The following lists the driver source code of MA35D1 GMAC interface:

- drivers/net/designware.c
- drivers/net/ma35d1_gmac.c

To enable GMAC support in U-Boot, please enable the following options in configuration interface.

```
[*] Networking support  --->
Device Drivers  --->
  [*] Network device support  --->
    [*]   Synopsys Designware Ethernet MAC
    [*]   Nuvoton MA35D1 GMAC
```

To use GMAC interface in U-Boot, a node that describes the attribute of GMAC needs to be present. Here describes the meaning of each attribute of GMAC.

The base address of GAMC is set to 40120000 for GMAC0 and 40130000 for GMAC1.

```
      gmac0: ethernet@40120000 {
```

“compatible” must be set to “nuvoton,ma35d1-gmac”.

```
compatible = "nuvoton,ma35d1-gmac";
```

“reg” defines the base address and size of GMAC control register. The base address is 0x40120000 for GAMC0 and 0x40130000 for GMAC1.

```
reg = <0x0 0x40120000 0x0 0x10000>;
```

The MAC address. Driver will use the MAC address stores in OTP if this entry is missing. In other words, this attribute can be removed if and only if and valid MAC address has been programmed into OTP.

```
mac-address = [ 00 11 22 33 44 55 ];
```

This attribute defines the PHY interface type, can be “rgmii-id” or “rmii”.

```
phy-mode = "rgmii-id";
```

Some setting of GMAC interface resides with system controller registers. GMAC driver gets the system controller base address through this attribute.

```
nuvoton,ma35d1-sys = <&sys>;
```

The following attribute is to set the MFP for GMAC interface. The pins used by GMAC is defined in pinctrl node.

```
pinctrl-names = "default";
pinctrl-0 = <&pinctrl_gmac0>;
```

The following attribute shows the PHY connected with GMAC only supports 10/100Mbps Ethernet for GMAC1.

```
phy-mode = "rmii";
pinctrl-0 = <&pinctrl_emac1>;
```

The following attribute is for driver to enable the GMAC clock setting. It is emac0_gate for GMAC0 and emac1_gate for GMAC1.

```
clocks = <&clk emac0_gate>;
clock-names = "stmmaceth";
```

The following attribute is for driver to reset the GAMC controller. It is MA35D1_RESET_GMAC0 for GMAC0 and MA35D1_RESET_GMAC1 for GMAC1.

```
resets = <&reset MA35D1_RESET_GMAC0>;
reset-names = "stmmaceth";
```

The GMAC id, set <0> for GMAC0 and <1> for GMAC1.

```
mac-id = <0>;
```

Set “okay” to enable GMAC controller, “disable” to disable GMAC controller.

```
status = "okay";
```

The following entries describe the attribute of PHY chip. phy-handle should point to a structure contains a single element that indicates the PHY address. This value has to set according to PHY address which is usually configurable by PHY’s power on setting. In the following example, it’s 1. compatible should be set to “snps, dwmac-mdio”. #address-cells and #size-cells should keep as it.

```
phy-handle = <&eth_phy0>;
mdio {
      compatible = "snps,dwmac-mdio";
```

```
                    #address-cells = <1>;
                    #size-cells = <0>;

                    eth_phy0: ethernet-phy@1 {
                          reg = <1>;
                    };
              };
        };
```

MA35D1 supports 2 GMAC and interface 0 is configured as default Ethernet interface. The environment variable “ethact” is used to select the default interface.

```
$ printenv ethact
ethact=ethernet@40120000
$ setenv ethact ethernet@40130000
$ print ethact
ethact=ethernet@40130000
```

3.6 USB

3.6.1 USB Host

MA35D1 USB host contains two types of host controller interfaces. One is EHCI that supports USB 2.0, the other is OHCI for USB 1.1 devices. The EHCI and OHCI driver for MA35D1 are listed below.

- drivers/usb/host/ohci-ma35d1.c
- drivers/usb/host/ehci-ma35d1.c

Below are the U-Boot options that has to be enable in order to support USB host controller. The “OHCI maximum root ports” should be set to 1.

```
Device Drivers  --->
  [*] USB support  --->
    [*]   Enable driver model for USB
    [*]   EHCI HCD (USB 2.0) support
    [*]     Support for Nuvoton MA35D1 on-chip EHCI USB controller
    [*]   OHCI HCD (USB 1.1) support
    (1)     OHCI maximum root ports
    [*]   Support for Nuvoton MA35D1 on-chip OHCI USB controller
```

Device tree uses different nodes to describe the attribute of EHCI and OHCI controllers.

This node specifies the EHCI register base address and clock gate.

```
        ehci0: ehci@40140000 {
              compatible = "nuvoton,ma35d1-ehci0";
              reg = <0x0 0x40140000 0x0 0x1000>;
              clocks = <&clk husbh0_gate>;
              status = "okay";
        };
```

This node specifies the OHCI register base address, clock gate and an extra element “nuvoton,sys” allowing OHCI to access system controller registers.

```
ohci0:ohci@40150000 {
        compatible = "nuvoton,ma35d1-ohci0";
        reg = <0x0 0x40150000 0x0 0x1000>;
        clocks = <&clk husbh0_gate>;
        nuvoton,sys = <&sys>;
        status = "okay";
};
```

3.7 Storage

3.7.1 SD card/eMMC

The following lists the MA35D1 driver source code to support SD controller:

- drivers/mmc/sdhci-dwcmshc.c

To enable the SD controller, the following U-Boot options have to be enabled.

```
Device Drivers  --->
  [*] Support block devices
  MMC Host controller Support  --->
    [*] MMC/SD/SDIO card support
    [*]   support for MMC/SD write operations
    [*] Enable MMC controllers using Driver Model
    [*] Secure Digital Host Controller Interface support
    [*]   SDHCI support for DesignWare Cores Mobile SDHCI controller
```

Here is an example of device tree node that describes the feature of MA35D1 SD controller.

```
sdhci0: sdhci@40180000 {
```

The following attributes configure the MFP setting of the pins used by SD controller.

```
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_sdhci0>;
```

“compatible” must be set to “snps,dwcmshc-sdhci”.

```
        compatible = "snps,dwcmshc-sdhci";
```

“reg” defines the base address and size of SD controller register. The base address is 0x40180000 for SD0 and 0x40190000 for SD1.

```
        reg = <0x0 0x40180000 0x0 0x2000>;
        interrupts = <GIC_SPI 30 IRQ_TYPE_LEVEL_HIGH>;
```

“clocks” setting defines the clock gating switch in clock control register. It is “sdh0_gate” for SD0 and “sdh1_gate” for SD1.

```
        clocks = <&clk sdh0_gate>;
```

“bus-width” defined the bus width of SD/eMMC interface, and can be set to 8 if SD0 interface is connected to an eMMC in 8-bit bus width, otherwise should be set to 4. “max-frequency” defines the maximum clock frequency for data transfer. SD driver will automatically select the highest clock

frequency that is no faster than this setting support by the storage device to do the data transfer.

```
            bus-width = <4>;
            max-frequency = <180000000>;
```

Set “status” to “okay” to enable SD controller, “disable” to disable SD controller

```
            status = "okay";
        };
```

3.7.2 NAND

The following lists the MA35D1 driver source code to support raw NAND:

- drivers/mtd/nand/raw/ma35d1_nand.c

To enable the NAND controller, the following U-Boot options have to be enabled.

```
Device Drivers  --->
  MTD Support  --->
    [*] Enable MTD layer
    [*] Enable Driver Model for MTD drivers
    [*] Raw NAND Device Support  --->
      [*]   Enable BBT (Bad Block Table) support
      [*]   MA35D1 NAND support
```

If accessing UBI volumes is required, the following U-Boot options also have to be enabled. Please note that it is recommend to set “UBI wear-leveling threshold” to 128 or 256 for MLC NAND flash. And the proper setting of “Maximum expected bad eraseblock count per 1024 eraseblocks” depends on the minimum and maximum NVM for the Flashes' endurance lifetime.

```
Device Drivers  --->
  MTD Support  --->
    UBI support  --->
      [*] Enable UBI - Unsorted block images
      (4096) UBI wear-leveling threshold
      (20)  Maximum expected bad eraseblock count per 1024 eraseblocks
```

The following is an example of device tree node that describes the feature of MA35D1 NAND controller.

```
        nand: nand@401A0000 {
```

The following attributes configures the MFP setting of the pins used by NAND controller.

```
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_nand>;
            u-boot,dm-pre-reloc;
```

“compatible” must be set to “nuvoton,ma35d1-nand”.

```
            compatible = "nuvoton,ma35d1-nand";
```

“reg” defines the base address and size of NAND controller register. The base address is 0x401A0000.

```
            reg = <0x0 0x401A0000 0x0 0x1000>;
```

NAND driver gets the resource through this string.

```
            reg-names = "nand";
```

This attribute is to allow WDT driver to get the system control register base address.

```
            nuvoton,sys = <&sys>;
```

"clocks" setting defines the clock gating switch in clock control register. And set "status" to "okay" to enable NAND controller, "disable" to disable NAND controller.

```
            clocks = <&clk nand_gate>;
            status = "okay";
```

The following attributes are related to the NAND flash feature and has to set accordingly.

- nand-ecc-mode: set the ECC mode and should be set to "hw_oob_first"
- nand-bus-width: defines the bus width and is always 8 for MA35D1.
- nand-ecc-strength: can be 8, 12, or 24. The value should be set according to the NAND Flash's requirement.
- nand-ecc-step-size: 512 for T8, T12 and is 1024 for T24
- nand-on-flash-bbt: tells uboot to use the on chip bad block table.

```
            nand-ecc-mode = "hw_oob_first";
            nand-bus-width = <8>;
            nand-ecc-strength = <8>;
            nand-ecc-step-size = <512>;
            nand-on-flash-bbt;
        };
```

3.8 Display

On the MA35D1 parts with LCD interface, the system can start showing images on the display while executing U-Boot, and does not have to wait until Linux kernel enabled frame buffer driver.

3.8.1 Display Interface

The following lists the MA35D1 driver source code to support Display interface:

- drivers/video/ma35d1_fb.c

To enable the Display interface, the following U-Boot options have to be enabled.

```
Device Drivers  --->
  Graphics Support  --->
  [*] Enable driver model support for LCD/video
    [*]   Nuvoton MA35D1 video support
```

Here is an example of device tree node describes the feature of Display interface.

```
        display: display@40260000 {
```

"compatible" must be set to "nuvoton,ma35d1-fb".

```
            compatible = " nuvoton,ma35d1-fb";
```

"reg" defines the base address and size of Display control register.

```
            reg = < 0x0 0x40260000 0x0 0x2000 >;
```

The following attribute is to set the MFP for Display interface. The pins used by Display is defined in pinctrl node.

```
pinctrl-names = "default";
pinctrl-0 = <&pinctrl_display0>;
```

The following attribute is for driver to enable the Display clock setting.

```
clocks = <&clk dcu_gate>;
```

The following attribute is for driver to reset the Display interface.

```
resets = <&reset MA35D1_RESET_DISP>;
```

Set “okay” to enable Display interface, “disable” to disable Display interface.

```
status = "okay";
```

“pixel-fmt” defines the framebuffer color format. This value must be set to “6”, it is A8R8G8B8.

```
pixel-fmt = <6>;
```

The following entries describe the attribute of LCD timing parameters. LCD timings should be created as nodes in the device tree with a display-timings subnode. These value has to set according to the own panel. In the following example, the native-mode is timing0.

“clock-frequency” defines the pixel clock in Hz.

“hactive”, “vactive” defines display resolution.

“hfront-porch”, “hback-porch”, “hsync-len” define horizontal display timing parameters in pixels.

“vfront-porch”, “vback-porch”, “vsync-len” define vertical display timing parameters in lines.

```
display-timings {
        native-mode = <&timing0>;
        timing0: 1024x600 {
                clock-frequency = <51000000>;
                hactive = <1024>;
                vactive = <600>;
                hsync-len = <1>;
                hfront-porch = <160>;
                hback-porch = <160>;
                vsync-len = <1>;
                vfront-porch  = <23>;
                vback-porch = <12>;
        };
        timing1: 800x480 {
                clock-frequency = <45000000>;
                hactive = <800>;
                vactive = <480>;
                hsync-len = <1>;
                hfront-porch = <210>;
                hback-porch = <46>;
                vsync-len = <1>;
```

```
                vfront-porch  = <22>;
                vback-porch = <23>;
        };
        timing2: 1920x1080 {
                clock-frequency = <125000000>;
                hactive = <1920>;
                vactive = <1080>;
                hsync-len = <32>;
                hfront-porch = <120>;
                hback-porch = <128>;
                vsync-len = <14>;
                vfront-porch  = <21>;
                vback-porch = <10>;
        };
};
```

MA35D1 Display interface supports splash screen. The image can be read into memory from a variety of storages. The following example demonstrates the image in the usb storage. To display image by command “bmp” after the picture preload to the DDR memory.

```
$ usb start
$ fatload usb 0 ${load DDR address} <bmp file name>
$ bmp d ${load DDR address}
```

3.8.2 BMP Logo display

To support BMP Logo Display during booting, update environment variables settings in ma35d1.h

- Include/configs/ma35d1.h

Add bmp_addr_r definition in CONFIG_EXTRA_ENV_SETTINGS that defines the loading address of BMP file. Since the display frame buffer of Linux Kernel is at 0x8c800000, we define the bmp_addr_r address to 0x8c800000.

```
/* Extra environment variables */
#define CONFIG_EXTRA_ENV_SETTINGS \
       "bootfile=Image\0" \
       "kernel_addr_r=0x80080000\0" \
       "fdt_addr_r=0x85000000\0" \
       "loadaddr=0x81000000\0" \
       "bmp_addr_r=0x8c800000\0" \
```

Besides, add two commands to MMCARGS definitions that executes automatically boot-up U-Boot.

The first command, “mmc read ${bmp_addr_r} 0xe000 0x1000”, is to load BMP file from SD card offset 28MB (0xe000 * 512) address. The read length is 2 MB (0x1000 * 512). If the BMP file is larger than 2MB, modify the read length. And remember to write the BMP file into SD card 28MB offset address before executing this command.

The second command, “bmp display ${bmp_addr_r}”, is to display BMP file to panel.

```
#define MMCARGS \
```

```
       "mmcboot=" \
       "if mmc dev 0; then " \
             "echo \"Booting form mmc ... ...\"; " \
             "setenv bootargs root=/dev/${mmc_block} rootfstype=ext4 rw rootwait
console=ttyS0,115200n8 rdinit=/sbin/init mem=${kernelmem}; " \
             "mmc read ${bmp_addr_r} 0xe000 0x1000; " \
             "bmp display ${bmp_addr_r}; " \
             "mmc read ${kernel_addr_r} 0x1800 0x8000; " \
             "mmc read ${fdt_addr_r} 0x1600 0x80; " \
```

Need to set framebuffer address of u-boot to the same as the kernel, and the default address of u-boot and kernel is at 0x8c800000, the following configurations should be set by menuconfig.

```
Device Drivers  --->
  Graphics Support  --->
  [*] Enable driver model support for LCD/video
    [*]   Nuvoton MA35D1 video support
       (0x8C800000) MA35D1 DCU framebuffer address
```

U-Boot provides a way to write BMP file into SD card. The following is an example to write a BMP file that is already downloaded to DDR address ${bmp_addr_r}. The first step is to erase SD card 28MB offset address, length 2MB. The second step is to write BMP file loaded at ${bmp_addr_r} to SD card 28MB offset. This ${bmp_addr_r} default is 0x8c800000, which is the reserved buffer for display in the kernel.

```
=> mmc erase 0xe000 0x1000
=> mmc write ${bmp_addr_r} 0xe000 0x1000
```

- arch/arm/dts/ma35d1-sdcard0.dts

In dts of u-boot, it defines memory length and default length is 0x07000000(112MB). However, we use 0x8c800000 for display BMP Logo and the address is over 0x87000000. Hence, we have to modify the memory length definition. The following is an example to enlarge memory length to 0x0f000000(240MB).

```
Memory {
       device_type = “memory”;
       reg = <0x00000000 0x80000000 0 0x0f000000>; /* 240M */
```

- arch/arm64/boot/dts/nuvoton/ma35d1-som-256m.dts

In dts of Linux kernel device tree file, the display need to remove “resets” to avoid display re-initialization.

```
&display {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_display0>;
        memory-region = <&display_buf>;
        //resets = <&reset MA35D1_RESET_DISP>;
        status = "okay";
};
```

3.9 Cryptographic

To prevent malicious user to change the binary images store in storage device, an encrypted form is preferable over the plan text form. However, booting an encrypted image takes longer due to the extra time used for decrypt the images. To overcome this issue, MA35D1 U-Boot supports AES decryption using hardware crypto accelerator.

Note that Cryptographic only supported on MA35D1 non-TSI chips.

3.9.1 AES

The driver and command source code supports AES decrypt function are:

- drivers/crypto/ma35d1-crypto.c
- cmd/nuvoton/nu_aes.c

To enable AES decrypt driver, the following U-Boot options need to be enabled.

```
Device Drivers  --->
  Hardware crypto devices  --->
    [*] Nuvoton MA35D1 cryptographic accelerator
Command line interface  --->
  Nuvoton specific command line interface  --->
    [*] Enable the command for AES decrypt image
```

Here is an example of device tree node that describes the feature of MA35D1 hardware AES accelerator.

```
      crypto: crypto@40300000 {
```

“compatible” must be set to “nuvoton,ma35d1-crypto”.

```
            compatible = "nuvoton,ma35d1-crypto";
```

“reg” defines the base address and size of NAND controller register. The base address is 0x40300000 for MA35D1.

```
            reg = <0x0 0x40300000 0x0 0x1000>;
```

AES driver gets the system control base address through this attribute. And set “status” to “okay” to enable AES accelerator, “disable” to disable AES accelerator.

```
            nuvoton,sys = <&sys>;
            status = "okay";
      };
```

The command to support AES decryption is as follow:

- KEY_NUM: The Key Store OTP key number which contains the AES key
- SRC: Address of the input cipher text.
- DST: Address to store the output plaintext. Could be the same with SRC.
- LEN: Length of the cipher text in bytes

```
$ nu_aes KEY_NUM SRC DST LEN
```

4 BUILDING U-BOOT

4.1 Build Command

To build U-Boot for MA35D1, a cross compiler needs to be assigned. User could use the following command to assign cross compiler and build U-Boot.

```
$ CROSS_COMPILE=aarch64-linux-gnu- make
```

Or simply use “make” to build U-Boot if CROSS_COMPILE is exported in advance.

```
$ export CROSS_COMPILE=aarch64-linux-gnu-
$ make
```

Make command can also specify the device tree blob to generate using DEVICE_TREE parameter while compiling U-Boot image.

```
$ CROSS_COMPILE=aarch64-linux-gnu- DEVICE_TREE=TARGET_DEVICE_TREE make
```

4.2 Generated Files

Below list some files will be generated after a successfully build.

- u-boot.bin

  U-Boot binary image with dtb append at the end.

- u-boot-nodtb.bin

  U-Boot binary image without dtb append at the end.

- u-boot.dtb

  Device tree blob files for U-Boot.

- u-boot

  U-Boot in executable file format, it is useful for debugging purpose.

- u-boot.map

  U-Boot image map file, it is useful for debugging purpose.

- u-boot.sym

  U-Boot symbol list it is useful for debugging purpose.

- System.map

  U-Boot symbol link address list, from low address to high address, it is useful for debugging purpose.

Please refer to NuWriter User Manual to load the generated images to target platform.

4.3 Cleanup Command

Three commands can be used for cleanup. The first command cleans minimum files, while the last command clean the most.

“make clean” remove most generated files but keep the config file.

```
$ make clean
```

“make mrproper” remove all generated files + config + various backup files

```
$ make mrproper
```

“make distclean” like “make mrproper” and also remove editor backup and patch files.

```
$ make distclean
```

5 BRING UP LINUX KERNEL

U-Boot can bring up Linux from different storage devices or from Ethernet. This chapter introduces the commands and environment variables to bring up Linux kernel from the devices supported by MA35D1.

These can be save as bootcmd environment variable and get execute automatically after system power up.

5.1 From NAND Flash

U-Boot supports booting from NAND Flash. Below is an example that reads Linux Kernel in NAND Flash offset 0x200000. Linux Kernel size is 10 MB and is loaded to DDR 0x80080000. Then, reading device tree blob to address 0x85000000 and using booti command to boot Linux Kernel.

```
U-Boot> nand read 0x80080000 0x200000 0xa00000
U-Boot> nand read 0x85000000 0x100000 0x10000
U-Boot> booti 0x80080000 - 0x85000000
```

5.2 From SPI NAND Flash

U-Boot supports booting from SPI NAND Flash. Below is an example that reads Linux Kernel in SPI NAND Flash offset 0x200000. Linux Kernel size is 10 MB and is loaded to DDR 0x80080000. Then, reading device tree blob to address 0x85000000 and using booti command to boot Linux Kernel.

```
U-Boot> sf probe
U-Boot> mtd read spi-nand0 0x80080000 0x200000 0xa00000
U-Boot> mtd read spi-nand0 0x85000000 0x100000 0x10000
U-Boot> booti 0x80080000 - 0x85000000
```

5.3 From SPI NOR Flash

U-Boot supports booting from SPI NOR Flash. Below is an example that reads Linux Kernel in SPI NOR Flash offset 0x200000. Linux Kernel size is 10 MB and is loaded to DDR 0x80080000. Then, reading device tree blob to address 0x85000000 and using booti command to boot Linux Kernel.

```
U-Boot> sf probe
U-Boot> sf read 0x80080000 0x200000 0xa00000
U-Boot> sf read 0x85000000 0x100000 0x10000
U-Boot> booti 0x80080000 - 0x85000000
```

5.4 From eMMC and SD Card

U-Boot supports booting from SD Card. Below is an example that reads Linux Kernel in SD Card offset 0x200000. Linux Kernel size is 12 MB and is loaded to DDR 0x80080000. Then, reading device tree blob to address 0x85000000 and using booti command to boot Linux Kernel.

```
U-Boot> mmc read 0x80080000 0x1000 0x6000
U-Boot> mmc read 0x85000000 0x200 0x80
U-Boot> booti 0x80080000 - 0x85000000
```

5.5 From Ethernet

U-Boot supports download image from Ethernet using TFTP protocol which is convenient during development stage. GMAC driver need to be enabled as well as network commands in order to use this feature.

To download image in a static IP environment, first set the IP address of GMAC interface and TFTP

server. In the following example 192.168.0.101 and 192.168.0.100 respectively. Then download the compressed Linux kernel image to address 0x80080000 and device tree blob to address 0x85000000 with “tftp” command.

```
U-Boot> setenv ipaddr 192.168.0.101
U-Boot> setenv serverip 192.168.0.100
U-Boot> tftp 85000000 ma35d1.dtb
U-Boot> tftp 82000000 Image.gz
```

To download load image in a dynamic IP environment, use “dhcp” command to acquire IP address from DHCP server and then use “tftp” command to download images.

```
U-Boot> setenv autoload no
U-Boot> dhcp
U-Boot> tftp 85000000 ma35d1.dtb
U-Boot> tftp 80080000 Image.gz
```

After download the kernel image, use “unzip” to decompress and “booti” to execute it. The command format is the same with other booting methods.

```
U-Boot> unzip 82000000 80080000
U-Boot> booti 80080000 - 85000000
```

5.6 FIT image

U-Boot also supports Flattened Image Tree (FIT) which is a new uImage structure which can include several images of various type into a single blob. Besides, FIT also supports compression, multiple images and configurations, and verified boot using SHA and RSA. The benefit for supporting multiple configuration is that a single FIT image can support different platforms. For example, on platform A, it uses the kernel image 1 and device tree blob 1 in the FIT image. And on platform B, it uses kernel image 1 and device tree blob2 in the same FIT image.

Please enable the following U-Boot options to support FIT image.

```
Boot images  --->
  [*] Support Flattened Image Tree
  [*]   Support SHA256 checksum of FIT image contents
  [*]   Enable signature verification of FIT uImages
  [*]     Support rsassa-pss signature scheme of FIT image contents
```

Here are the commands to create a new RSA2048 private key and its x509 certificate.

```
$ openssl genpkey -algorithm RSA -out keys/dev.key -pkeyopt rsa_keygen_bits:2048
$ openssl req -batch -new -x509 -key keys/dev.key -out keys/dev.crt
```

Here is an example of the FIT description file. This file is actually a device tree source file containing the information of the files to be pack together and the supported configuration. Please note that some “description” fields are mandatory, the tool for packing FIT image, mkimage, reports error if these fields are missing.

```
/dts-v1/;

/ {
      description = “Linux kernel with one dtb”
      #address-cells = <1>;
```

In this configuration, one kernel image and one device tree blob are added. SHA256 is assigned as the hash algorithm. Kernel image is stored in compressed format while device tree blob stored in plaintext format.

```
images {
	kernel {
		description = "Linux kernel image”;
		data = /incbin/("Image.gz");
		type = "kernel";
		arch = "arm64";
		os = "linux";
		compression = "gzip";
		load = <0x80080000>;
		entry = <0x80080000>;
		hash-1 {
			algo = "sha256";
		};
	};
	fdt-1 {
		description = “Device tree blob”
		data = /incbin/("ma35d1.dtb");
		type = "flat_dt";
		arch = "arm64";
		compression = "none";
		load = <0x88000000>;
		hash-1 {
			algo = "sha256";
		};
	};
};
```

While FIT supports multiple configurations, this example only assign a single configuration conf-1, and the images for this configuration are signed using RSA2048.

```
configurations {
	default = "conf-1";
	conf-1 {
		description = “Default configuration”;
		kernel = "kernel";
		fdt = "fdt-1";
		signature {
			algo = "sha256,rsa2048";
			key-name-hint = "dev";
			sign-images = "fdt", "kernel";
```

```
                    };
                };
        };
};
```

Use the “mkimage” under tools directory to generate a signed blob, image.fit in the following sample.

- -f: Assign the input filename for FIT source
- -K: Write public keys to this.dtb file
- -k: Set directory containing private keys
- -r: Mark keys used as 'required' in dtb

```
$ tools/mkimage -f images.its -K u-boot.dtb -k keys/ -r image.fit
```

Repack the U-Boot image with the dtb file containing the public key.

```
$ cat u-boot-nodtb.bin u-boot.dtb >u-boot-dtb.bin
```

After booting into U-Boot command line interface, an “iminfo” command follow by the FIT image address can be used to check the header information of the FIT.

```
$ iminfo 82000000
```

To boot the FIT image, use “boom” command follow by the FIT image address, a hash sign, and the configuration to use.

```
$ bootm 82000000#conf-1
```

If the signature and hash are not required, the hash-1 and signature fields in previous sample can be omitted. Below is the command for generate unsigned FIT image, also there is no need to repack U-Boot dtb file with U-Boot binary image.

```
$ tools/mkimage -f images.its image.fit
```

The command for launching the configuration without signature is the same with the command to launch a signed configuration.

6 REVISION HISTORY

Date	Revision	Description
2022.11.23	1.02	Modify the description in the document to match the actual status of the BSP.
2022.08.04	1.01	Add section 3.8.2 BMP logo display
2022.03.10	1.00	Preliminary issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.