Aspeed Zephyr
SDK User Guide

Version 00.01.05

ASPEED Technology Inc.

Table of Content

Revisions 3

1. Overview 4

2. Licenses 4

3. SDK build 4

   3.1. Zephyr Environment Setup 4

   3.2. Install a toolchain 5

   3.3. Configurations 6

   3.4. Execute the SDK image on the Aspeed EVB..8

4. Operation system 18

5. Multi-function pin 19

   5.1. Driver configure flow 19

6. Drivers and Demo APPs..21

   6.1. Aspeed EVB Demo sample 21

   6.2. GPIO/SGPIO Driver 24

   6.3. ADC Driver 27

   6.4. PECI Driver..30

   6.5. JTAG master Driver 33

   6.6. PWM/TACH Driver..35

   6.7. eSPI Driver 40

   6.8. I2C Driver..41

   6.9. IPMI KCS Driver 48

   6.10. IPMI BT Driver 50

   6.11. USB Device Driver 51

   6.12. UART Driver..55

   6.13. Virtual UART Driver 57

   6.14. Snoop Driver 59

   6.15. PCC Driver 60

   6.16. Mailbox Driver..62

   6.17. FMC_SPI Driver 63

   6.18. I3C driver 69

   6.19. Crypto driver 80

   6.20. Watchdog Timer (WDT) 86

   6.21. Timer driver..91

7. Debug 95

   7.1. UART console 95

   7.2. Hardware debug UART...95

7.3. JTAG ICE 96
8. Alternate Bootblock Recovery (ABR) 97
8.1. FMC ABR with Two Flashes 98
8.2. FMC ABR with Single Flash..102
8.3. FW Update with FMC ABR Enabled 106
8.4. SPI1 ABR with Two Flashes 106
8.5. SPI1 ABR with Single Flash 109
8.6. Host Image Update with SPI1 ABR Enabled..111
9. One Time Programmable Memory 113
9.1. OTP Memory Organization...113
9.2. OTP tool...117
9.3. OTP Utility 119
10. Secure Boot...122
10.1. Root of Trust (RoT) 122
10.2. RoT Boot Mode 124
10.3. SOC Secure Tool 127
11. Reference 130

Revisions

Version	Description
00.01.00	Initial create
00.01.01	Add FMC_SPI, I3C and HACE drivers
00.01.02	Update for tag v00.01.02 release
00.01.03	Update for tag v00.01.03 release
00.01.04	Update for tag v00.01.04 release
00.01.05	Update for tag v00.01.05 release

1. Overview

This document provides the information for using Aspeed Zephyr SDK on the following Aspeed products:

- AST1030 series SOC: Bridge IC (BIC)
- AST2600 series SOC: Secondary service processor (SSP)

Aspeed Zephyr SDK is forked from the zephyrproject/zephyr-rtos [1], plus Aspeed device drivers. The source files can be downloaded from Aspeed GitHub [2].

2. Licenses

Aspeed Zephyr SDK is licensed using the Apache 2.0 license, as descripted in the Zephyr Project Documentation [3].

3. SDK build

This chapter describes how to cross-compile and generate the Zephyr binary image on the host machine.

3.1. Zephyr Environment Setup

The SDK develop environment is based on Ubuntu 18.04 LTS – 64bits and bash shell. Also, the tools listed in the following sections must be installed on the host machine. See Zephyr Project Document [3] for more details.

3.1.1. Install dependencies

Please issue the following command for installing required packages:

```
#sudo apt install --no-install-recommends git cmake ninja-build gperf \
  ccache dfu-util device-tree-compiler wget \
  python3-dev python3-pip python3-setuptools python3-tk python3-wheel xz-
utils file \
  make gcc gcc-multilib g++-multilib libsdl2-dev
```

Verify the versions of the main dependencies installed on your system:

```
# cmake --version
# python3 --version
# dtc --version
```

Minimum required.

name	version
CMake	3.13.1
Python	3.6
Devicetree compiler	1.4.6

3.1.2. Download Aspeed Zephyr package

Install west, and make sure ~/.local/bin is on your PATH environment variable:

```
# pip3 install --user -U west
# echo 'export PATH=~/.local/bin:"$PATH"' >> ~/.bashrc
# source ~/.bashrc
```

Get the Aspeed Zephyr BSP from GitHub:

```
# west init -m https://github.com/AspeedTech-BMC/zephyr.git --mr
v00.01.05 zephyrproject
# cd zephyrproject
# west update
```

Zephyr’s scripts/requirements.txt file declares additional Python dependencies. Install them with pip3.:

```
# pip3 install --user -r ~/zephyrproject/zephyr/scripts/requirements.txt
```

3.2. Install a toolchain

Download the latest SDK installer:

```
# cd ~
# wget https://github.com/zephyrproject-rtos/sdk-
ng/releases/download/v0.12.4/zephyr-sdk-0.12.4-x86_64-linux-setup.run
```

Run the installer, installing the SDK in ~/zephyr-sdk-0.12.4:

```
# chmod +x zephyr-sdk-0.12.4-x86_64-linux-setup.run
# ./zephyr-sdk-0.12.4-x86_64-linux-setup.run -- -d ~/zephyr-sdk-0.12.4
```

Setup the environment variables:

```
# touch ~/.zephyrrc
# echo "export ZEPHYR_TOOLCHAIN_VARIANT=zephyr" > ~/.zephyrrc
# echo "export ZEPHYR_SDK_INSTALL_DIR=/home/"$(whoami)"/zephyr-sdk-
0.12.4" >> ~/.zephyrrc
```

3.3. Configurations

Aspeed Zephyr SDK uses Kconfig system to manage the configurations among the drivers, applications, and kernel operation system.

3.3.1. Default configuration file

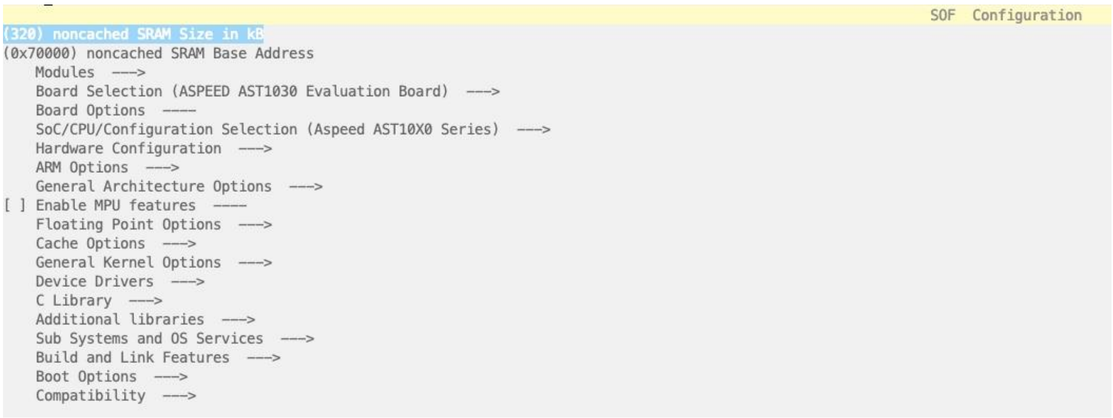
The default configuration file defines the default properties of the board target. All the default configuration files are placed in boards folder.

Targets	location
ASPEED AST2600 EVB	boards/arm/ast2600_evb/ast2600_evb_defconfig
ASPEED AST1030 EVB	boards/arm/ast1030_evb/ast1030_evb_defconfig

Here we take ast1030_evb and hello_world application for example:

```
# source zephyr-env.sh
# west build -b ast1030_evb -t menuconfig samples/hello_world
```

Modify the configurations if necessary



```
                                                                SOF  Configuration
(320) noncached SRAM Size in kB
(0x70000) noncached SRAM Base Address
    Modules  --->
    Board Selection (ASPEED AST1030 Evaluation Board)  --->
    Board Options  ----
    SoC/CPU/Configuration Selection (Aspeed AST10X0 Series)  --->
    Hardware Configuration  --->
    ARM Options  --->
    General Architecture Options  --->
[ ] Enable MPU features  ----
    Floating Point Options  --->
    Cache Options  --->
    General Kernel Options  --->
    Device Drivers  --->
    C Library  --->
    Additional libraries  --->
    Sub Systems and OS Services  --->
    Build and Link Features  --->
    Boot Options  --->
    Compatibility  --->
```

Next, build application

```
# west build -b ast1030_evb samples/hello_world
```

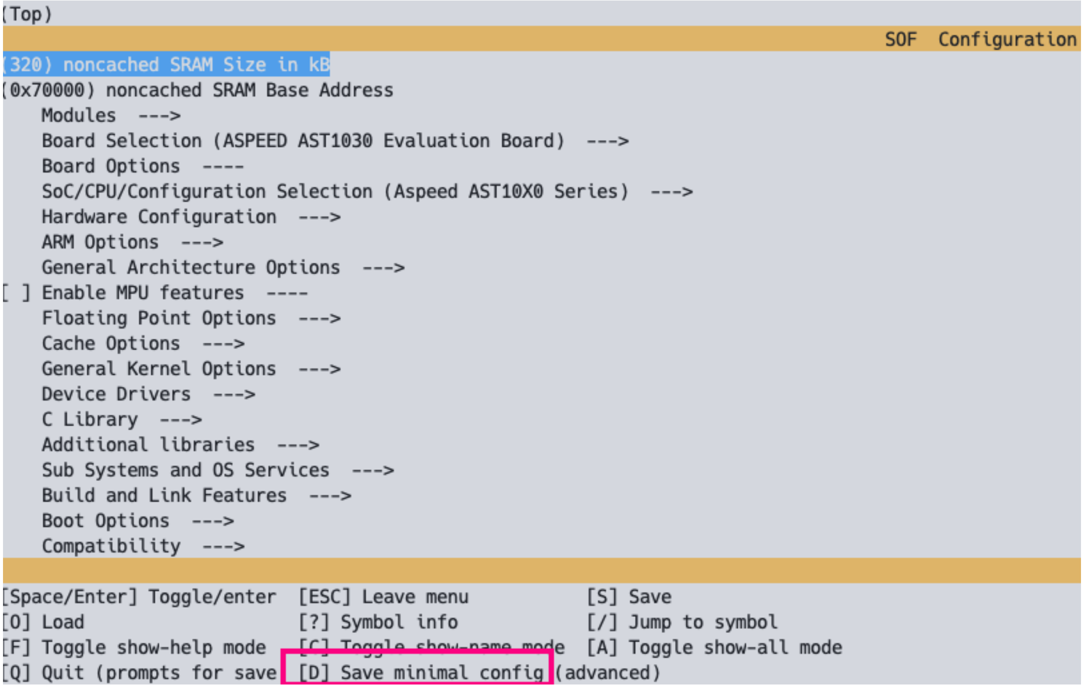
And the target files “`zephyr.bin`” will be generated in `build/zephyr` folder.   For UART boot you need to use “`uart_zephyr.bin`”.

3.3.2. Add customized configuration file

Create a new defconfig file from an exist one (e.g., ast1030_evb_defconfig).

```
# west build -b ast1030_evb -t menuconfig samples/hello_world
```

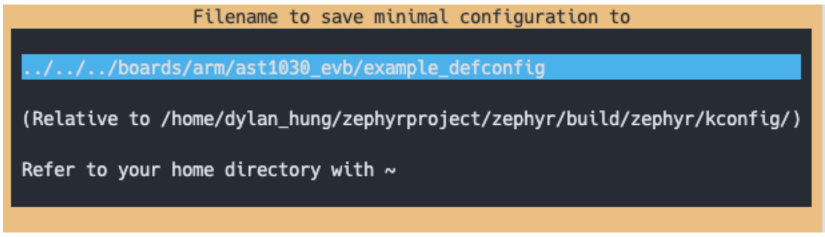
Modify the configurations if necessary.   Then, press “D” to save the configuration



```
(Top)
                                                                  SOF  Configuration
(320) noncached SRAM Size in kB
(0x70000) noncached SRAM Base Address
    Modules  --->
    Board Selection (ASPEED AST1030 Evaluation Board)  --->
    Board Options  ----
    SoC/CPU/Configuration Selection (Aspeed AST10X0 Series)  --->
    Hardware Configuration  --->
    ARM Options  --->
    General Architecture Options  --->
[ ] Enable MPU features  ----
    Floating Point Options  --->
    Cache Options  --->
    General Kernel Options  --->
    Device Drivers  --->
    C Library  --->
    Additional libraries  --->
    Sub Systems and OS Services  --->
    Build and Link Features  --->
    Boot Options  --->
    Compatibility  --->

[Space/Enter] Toggle/enter  [ESC] Leave menu           [S] Save
[O] Load                    [?] Symbol info            [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

Enter the name for your customized defconfig



```
                 Filename to save minimal configuration to

 ../../../boards/arm/ast1030_evb/example_defconfig

 (Relative to /home/dylan_hung/zephyrproject/zephyr/build/zephyr/kconfig/)

 Refer to your home directory with ~
```

3.4. Execute the SDK image on the Aspeed EVB

3.4.1. AST1030 EVB

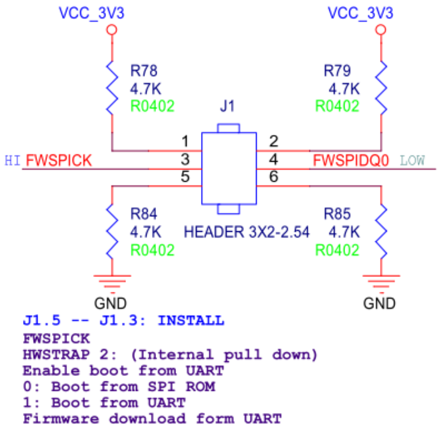
There are two boot modes for AST1030 EVB: one is from UART and the other is from embedded SPI Flash memory.

3.4.1.1. UART boot

This section demonstrates how to load the SDK image by AST1030 HW boot-from-UART function.

3.4.1.1.1. Prerequisite

- Prepare the image "`uart_zephyr.bin`". This image file will be generated in `zephyr/build/` folder.
- Set the strap to "BOOT UART" by connecting `FWSPICK` to VCC_3V3



J1.5 -- J1.3: INSTALL
FWSPICK
HWSTRAP 2: (Internal pull down)
Enable boot from UART
0: Boot from SPI ROM
1: Boot from UART
Firmware download form UART

- Connect EVB UART5 with your PC COM port.
- Open `ast1030_uart_download.ttl` in the text editor and modify the variable `filename` to your local path.

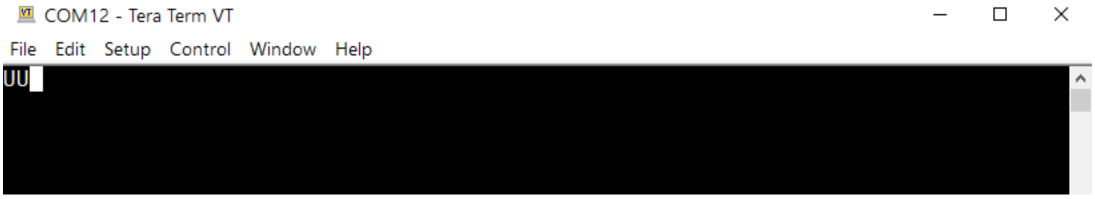
  For example: `filename = 'D:\tmp\uart_zephyr.bin'`

  The ttl script is appended in the SDK with the following path

source files	`boards/arm/ast1030_evb/tools/ast1030_uart_download.ttl`
environment	Host: tera-term script executes on Windows PC EVB: HW UART boot prompt

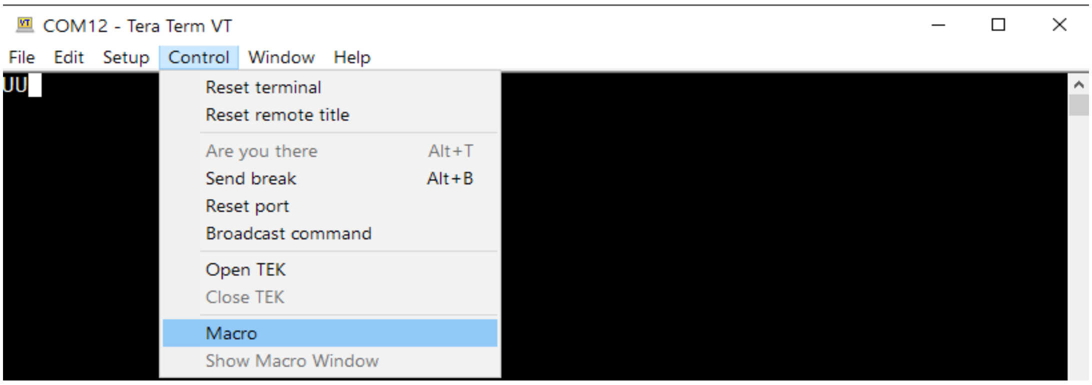
3.4.1.1.2. Loading the image via UART5

Power on the AST1030 EVB. Two "U" characters will be printed out by the AST1030 SOC.



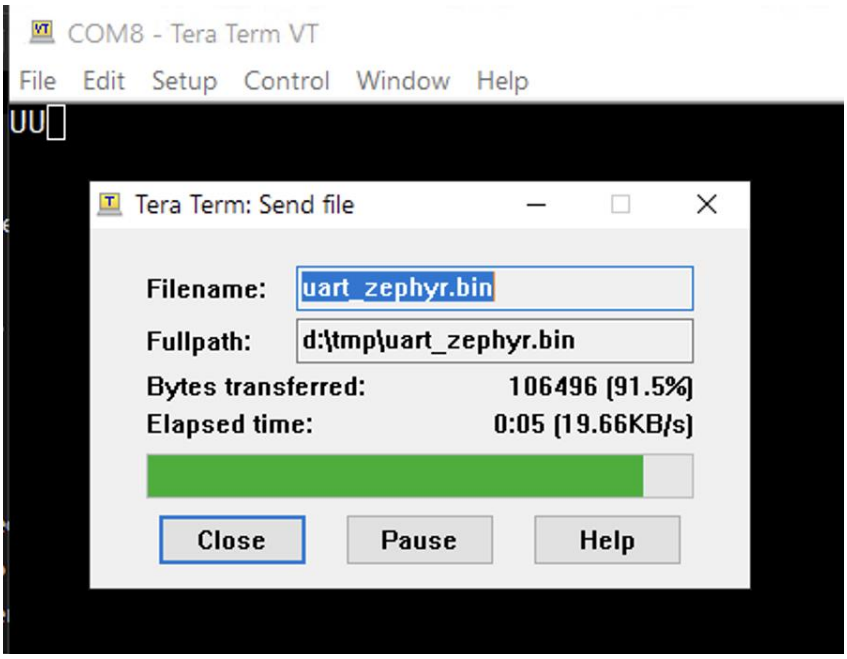
```
UU
```

Select "Control" → "Macro"



```
UU
```

Select the path of your macro file `ast1030_uart_download.ttl`. Then AST1030 will successfully boot once the image is loaded.



```
UU
```

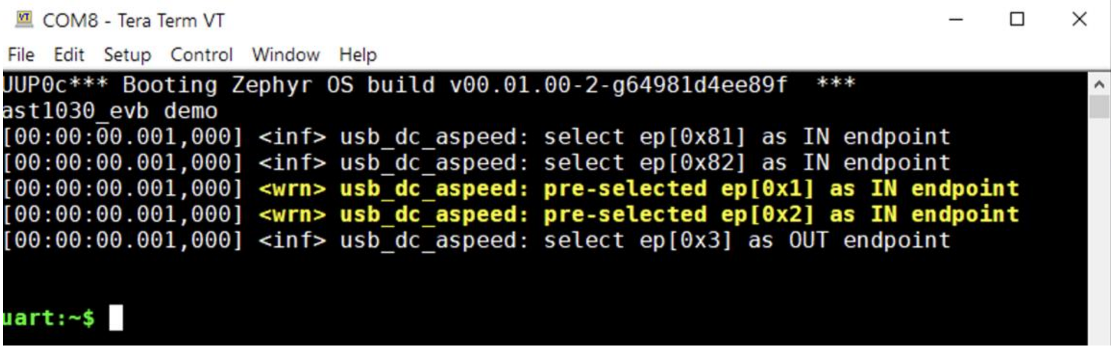
Filename: uart_zephyr.bin

Fullpath: d:\tmp\uart_zephyr.bin

Bytes transferred: 106496 (91.5%)

Elapsed time: 0:05 (19.66KB/s)

Finish



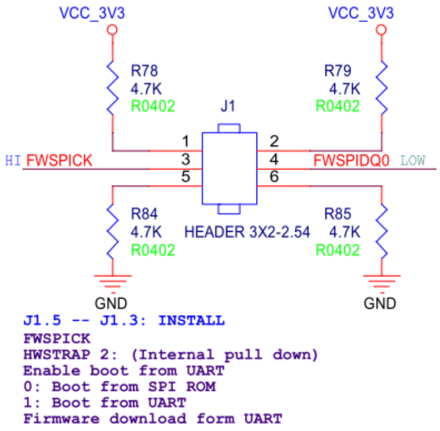
```
UUP0c*** Booting Zephyr OS build v00.01.00-2-g64981d4ee89f  ***
ast1030_evb demo
[00:00:00.001,000] <inf> usb_dc_aspeed: select ep[0x81] as IN endpoint
[00:00:00.001,000] <inf> usb_dc_aspeed: select ep[0x82] as IN endpoint
[00:00:00.001,000] <wrn> usb_dc_aspeed: pre-selected ep[0x1] as IN endpoint
[00:00:00.001,000] <wrn> usb_dc_aspeed: pre-selected ep[0x2] as IN endpoint
[00:00:00.001,000] <inf> usb_dc_aspeed: select ep[0x3] as OUT endpoint

uart:~$ █
```

3.4.1.2. SPI Flash boot

3.4.1.2.1. Prerequisite

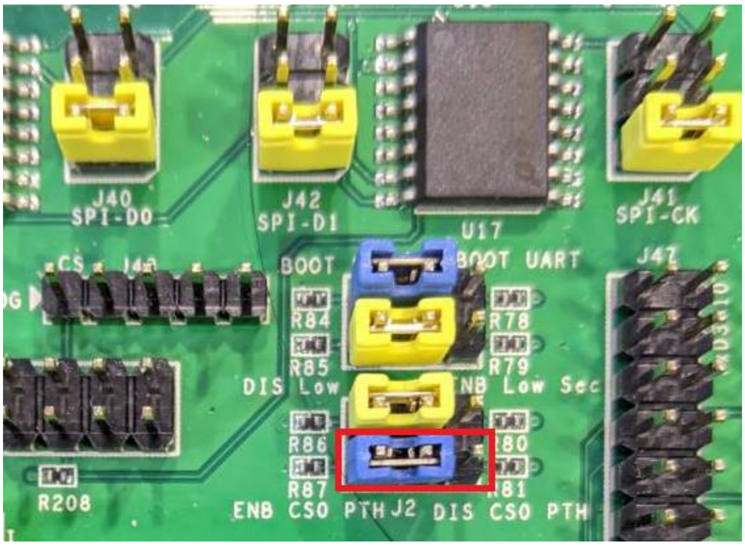
- Set the strap to “BOOT SPI” by connecting FWSPICK to GND



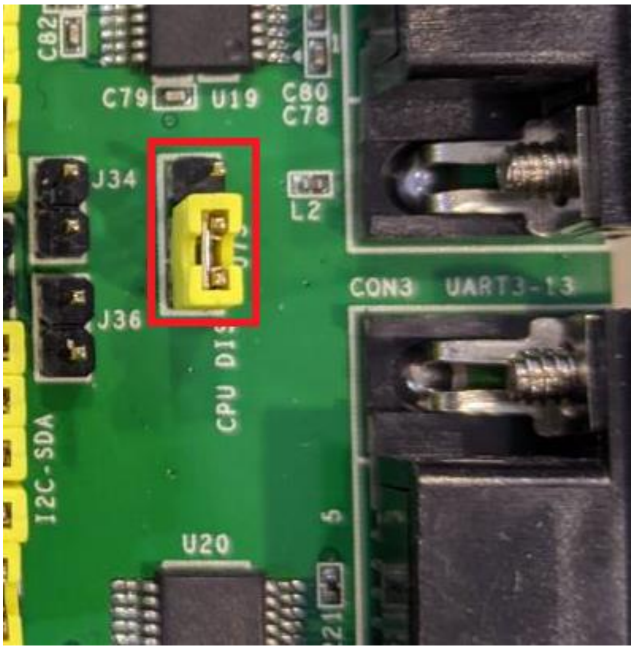
3.4.1.3. Download image by SF-100 programmer on AST1030 EVB

3.4.1.3.1. AST1030-A0:

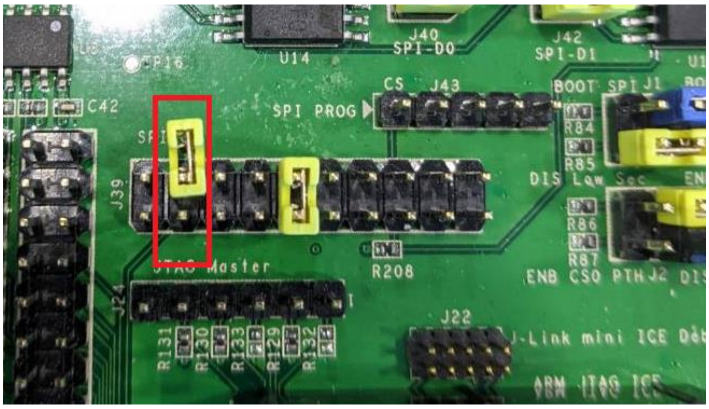
- Enable FMC CS0 path through mode by connect J2 pin 4 and pin 6.

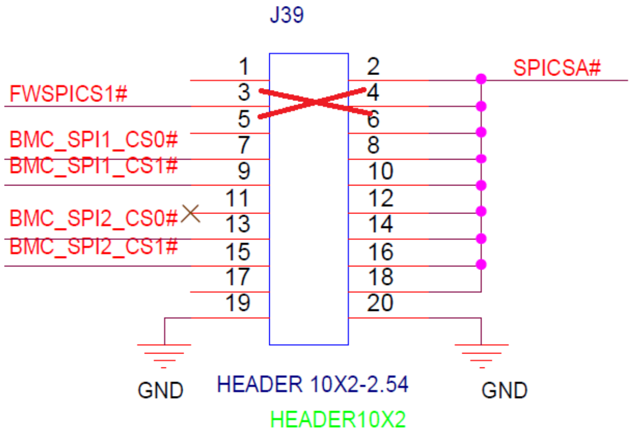


- Disable CPU by connect J75 pin1 and pin 2.



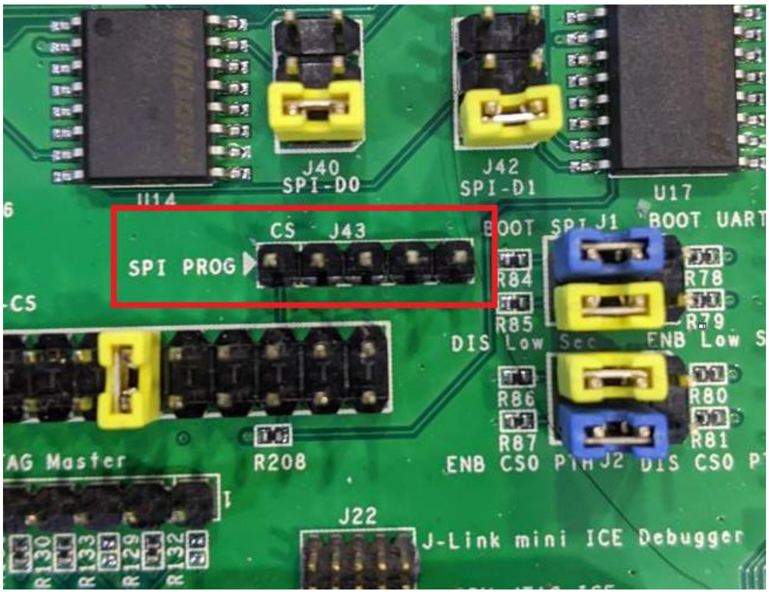
- Check J39 J3 and J4 is disconnected.



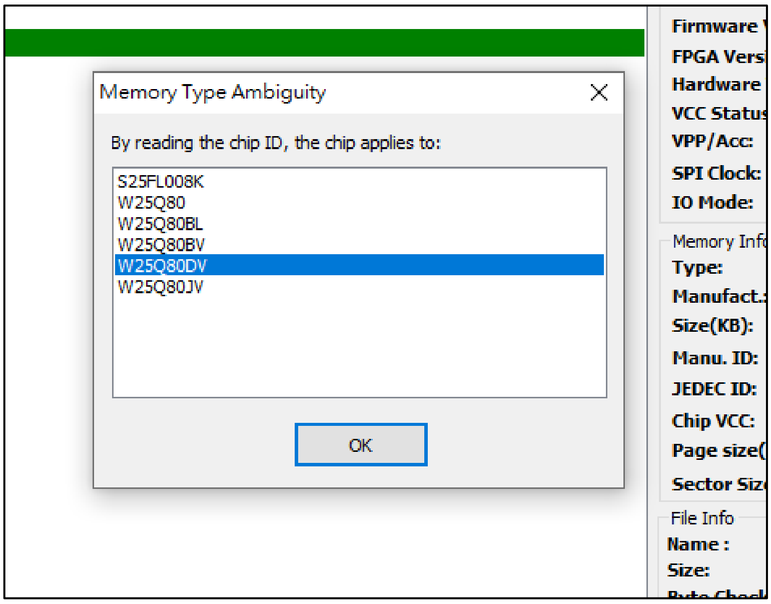


- Connect SF-100 programmer

J43 pin1 => CS	(SF100#7 CS)
J43 pin2 => CLK	(SF100#8 CLK)
J43 pin3 => MOSI	(SF100#10 MOSI)
J43 pin4 => MISO	(SF100#9 MISO)
J43 pin5 => GND	(SF100#6 GND)



- Power on device and select W25Q80DV for AST1030-A0



Memory Type Ambiguity

By reading the chip ID, the chip applies to:

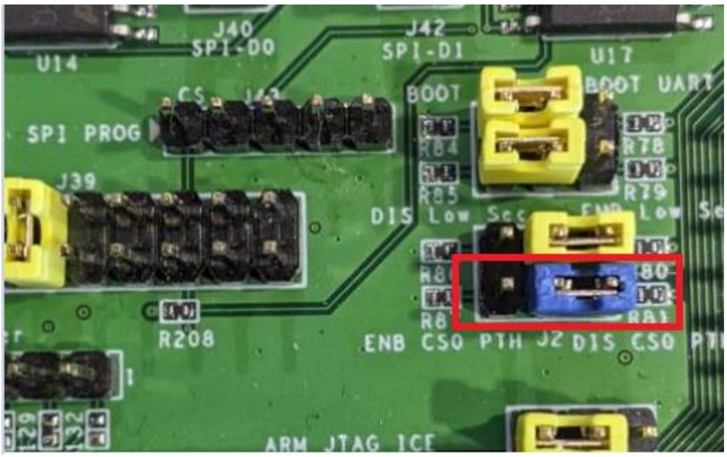
- S25FL008K
- W25Q80
- W25Q80BL
- W25Q80BV
- W25Q80DV
- W25Q80JV

OK

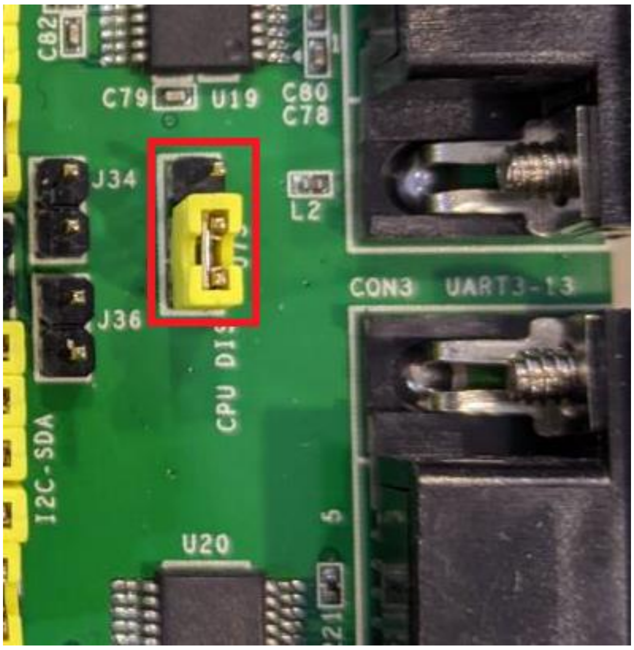
- Select your file and download it into the flash.
- **After download process is finished, remember to restore the jumper position of J2 and J75 for disabling CS0 path through and enable CPU.**

3.4.1.3.2. AST1030-A1:

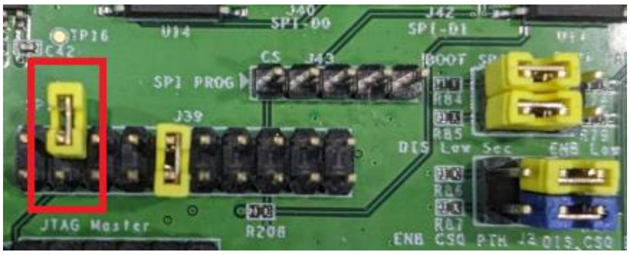
- Enable FMC CS0 path through mode by connect J2 pin 2 and pin 4.

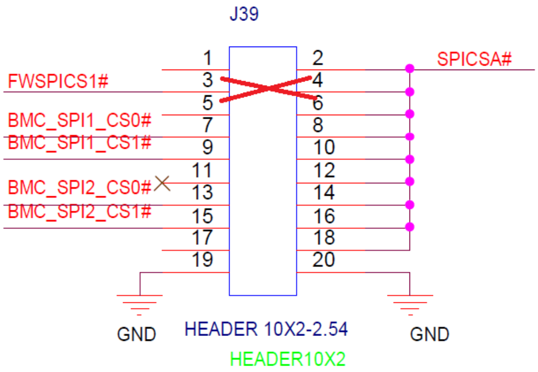


- Disable CPU by connect J75 pin1 and pin 2.



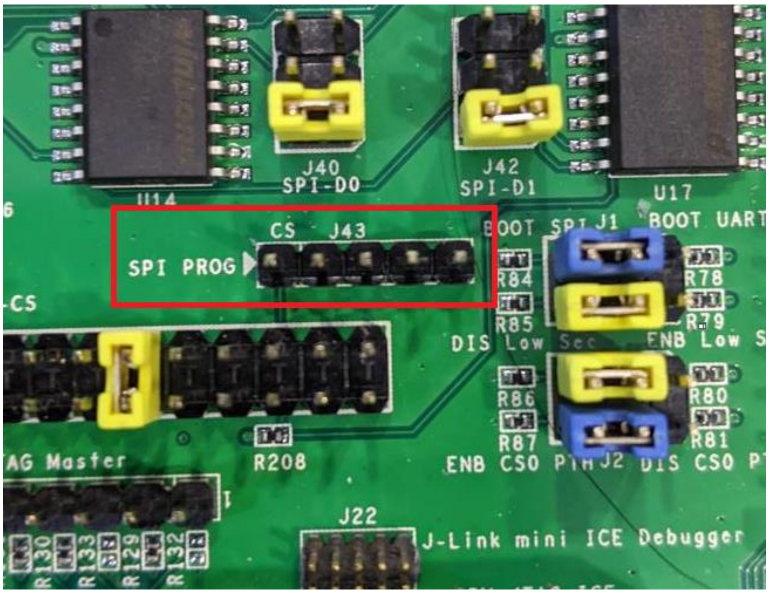
- Check J39 J3 and J4 is disconnected.



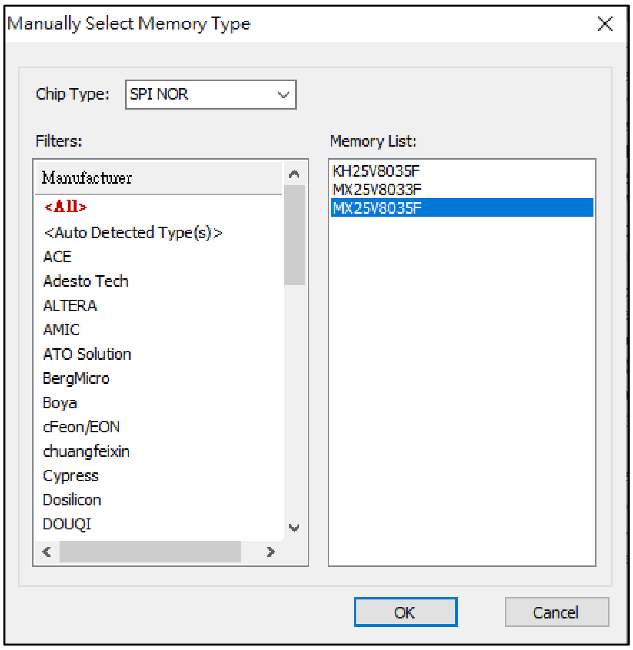


- Connect SF-100 programmer

J43 pin1 => CS	(SF100#7 CS)
J43 pin2 => CLK	(SF100#8 CLK)
J43 pin3 => MOSI	(SF100#10 MOSI)
J43 pin4 => MISO	(SF100#9 MISO)
J43 pin5 => GND	(SF100#6 GND)



- Power on device and select MX25V8035F for AST1030-A1


Manually Select Memory Type

Chip Type: SPI NOR

Filters:

Memory List:
KH25V8035F
MX25V8033F
MX25V8035F

- Select your file and download it into the flash.
- **After download process is finished, remember to restore the jumper position of J2 and J75 for disabling CS0 path through and enable CPU.**

3.4.2. AST2600 EVB

Execute the tera-term macro to load and execution the Zephyr SDK image on the AST2600 EVB

source files	`boards/arm/ast2600_evb/tools/ast2600_uart_download.ttl`
environment	Host: Tera-term script executes on Windows PC.

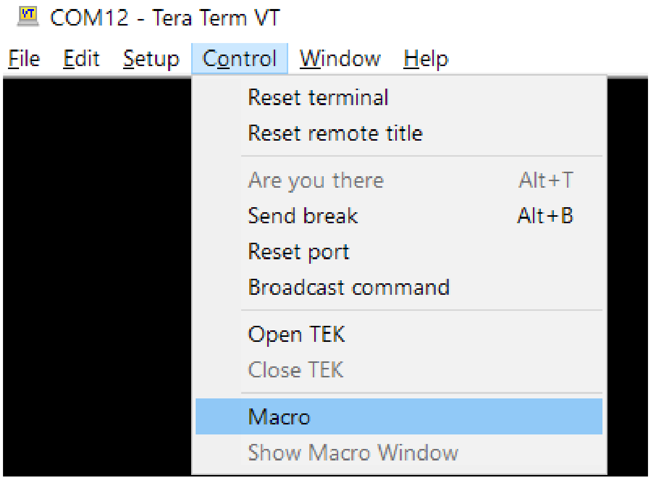
	EVB: AST2600 u-boot console

3.4.2.1. Prerequisite

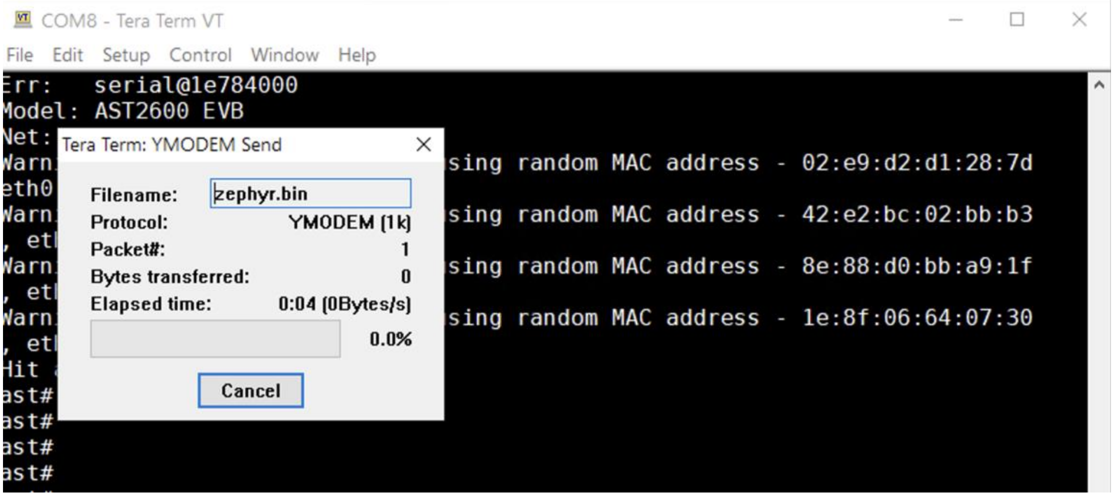
- Open `ast2600_uart_download.ttl` in the text editor and modify the variable `filename` to your local path.
  For example: `filename = ‘D:\tmp\ast2600_ssp.bin’`
- Ensure the primary service processor (Cortex-A7) stays in u-boot prompt.
- Connect EVB UART5 (primary service processor UART console) and UART11 (secondary service processor UART console) with your PC COM port

3.4.2.2. Loading the image via YMODEM

Select “Control” → “Macro”



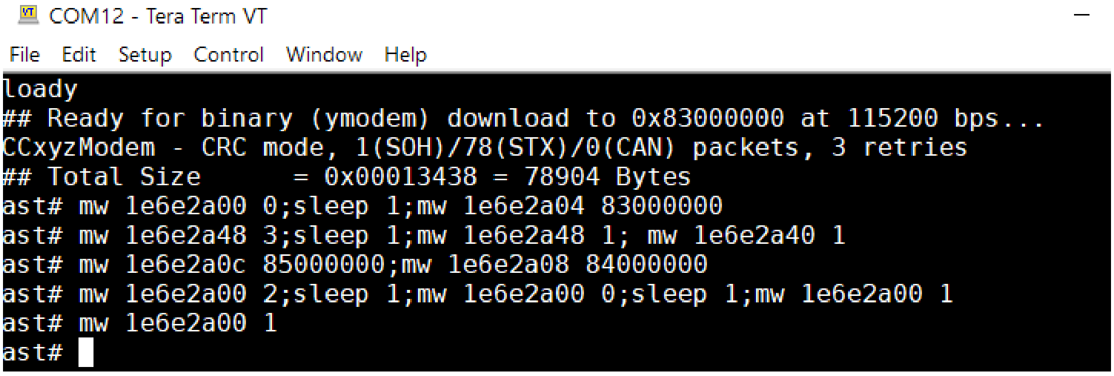
Select the path of your local macro file. Then the image file will be loaded onto the AST2600 EVB.



```
Err:   serial@1e784000
Model: AST2600 EVB
Net:
Warn                         sing random MAC address - 02:e9:d2:d1:28:7d
eth0
Warn                         sing random MAC address - 42:e2:bc:02:bb:b3
, et
Warn                         sing random MAC address - 8e:88:d0:bb:a9:1f
, et
Warn                         sing random MAC address - 1e:8f:06:64:07:30
, et
Hit
ast#
ast#
ast#
ast#
```

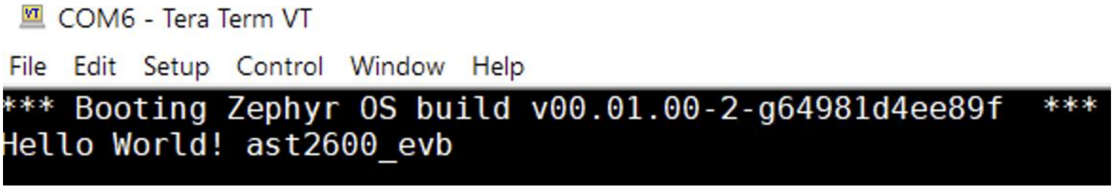
Filename: zephyr.bin
Protocol: YMODEM (1k)
Packet#: 1
Bytes transferred: 0
Elapsed time: 0:04 (0Bytes/s)
0.0%

Finish.



```
loady
## Ready for binary (ymodem) download to 0x83000000 at 115200 bps...
CCxyzModem - CRC mode, 1(SOH)/78(STX)/0(CAN) packets, 3 retries
## Total Size      = 0x00013438 = 78904 Bytes
ast# mw 1e6e2a00 0;sleep 1;mw 1e6e2a04 83000000
ast# mw 1e6e2a48 3;sleep 1;mw 1e6e2a48 1; mw 1e6e2a40 1
ast# mw 1e6e2a0c 85000000;mw 1e6e2a08 84000000
ast# mw 1e6e2a00 2;sleep 1;mw 1e6e2a00 0;sleep 1;mw 1e6e2a00 1
ast# mw 1e6e2a00 1
ast#
```

Check AST2600 EVB UART11 output



```
*** Booting Zephyr OS build v00.01.00-2-g64981d4ee89f  ***
Hello World! ast2600_evb
```

4. Operation system

Zephyr kernel version v2.6.0 [4] is used in Aspeed Zephyr SDK. For convenience and ease of use, Aspeed Zephyr SDK enables CMSIS RTOS API by default. The users can turn it off by Kconfig:

interface	`zephyr/include/portability/cmsis_os.h` `zephyr/include/portability/cmsis_os2.h`
Kconfig source files	`zephyr/subsys/portability/cmsis_rtos_v2/Kconfig`

The CMSIS RTOS document can be found in the Keil CMSIS RTOS2 website.
https://www.keil.com/pack/doc/CMSIS/RTOS2/html/index.html
Limitations of using os wrapper of CMSIS RTOS2 in zephyr:
https://docs.zephyrproject.org/latest/guides/portability/cmsis_rtos_v2.html

5. Multi-function pin

5.1. Driver configure flow

5.1.1. Add signal

Soc name	file
AST2600	`soc/arm/aspeed/ast26xx/sig_def_list.h`
AST1030	`soc/arm/aspeed/ast10x0/sig_def_list.h`

sig_def_list.h describes the pin used and scu setting for the signal.

For example:

```
/*
* ADC0 use ball E3 and need to set scu430[24] = 0
* ADC1 use ball D1 and need to set scu430[25] = 0
*/
SIG_DEFINE(ADC0, E3, SIG_DESC_CLEAR(0x430, 24))
SIG_DEFINE(ADC1, D1, SIG_DESC_CLEAR(0x430, 25))
```

5.1.2. Add function dts node

Soc name	file
AST2600	`dts/arm/aspeed/ast26xx-pinctrl.dtsi`
AST1030	`dts/arm/aspeed/ast10x0-pinctrl.dtsi`

The astxxxx-pinctrl.dtsi describes the function symbol used in dts.

For example:

```
&pinmux {
    pinctrl_adc0_default: adc0_default {};
    pinctrl_adc1_default: adc1_default {};
…
};
```

5.1.3. Combine function with signal

board name	file
`ast2600_evb`	`boards/arm/ast2600_evb/fun_def_list.h`
`ast1030_evb`	`boards/arm/ast1030_evb/fun_def_list.h`

fun_def_list.h describes the signal used for the function.

For example:

```
/*
* The function pinctrl_adc0_default uses signal ADC0
* The function pinctrl_adc1_default uses signal ADC1
*/
FUN_DEFINE(DT_NODELABEL(pinctrl_adc0_default), ADC0)
FUN_DEFINE(DT_NODELABEL(pinctrl_adc1_default), ADC1)
```

5.1.4. Add pinctrl-0 property to yaml

Each aspeed device need to add below to device binding for pinctrl-0 property:

```
include: [aspeed-pinctrl.yaml]
```

5.1.5. Usage

board name	file
ast2600_evb	boards/arm/ast2600_evb/ast2600_evb.dts
ast1030_evb	boards/arm/ast1030_evb/ast1030_evb.dts

Enable the device and the pinmux used.

For example:

```
&adc0 {
    status = "okay";
    pinctrl-0 = <&pinctrl_adc0_default &pinctrl_adc1_default>;
};
```

5.1.6. Debug

When the log message below is found:

```
<err> pinmux_aspeed: pin 76 already occupied by 73
```

It means the pinctrl settings conflicts with another pins. For debug, we can enable PINCTRL_STRING_NAME through menuconfig

```
(Top) → Device Drivers → Enable board pinmux driver → Aspeed SOC
pinmux driver  [*] Debug: print pin/signal with string name
```

After that, the log becomes easy to understand which pins are in conflicts.

```
<err> pinmux_aspeed: pin C19 already occupied by HVI3C5SC
```

6. Drivers and Demo APPs

Zephyr build system is application-centric and requires Zephyr-based applications to initiate building the kernel source tree. The application build controls the configuration and build process of both the application and Zephyr itself, compiling them into a single binary. There are many samples under `zephyr/samples` that the develops can refer to. The following chapters detail the how to enable and configure the drivers and how the driver works with the sample applications.

6.1. Aspeed EVB Demo sample

Aspeed Zephyr SDK provides a sample application which enables most of the device drivers for Aspeed SOCs.

6.1.1. AST2600-EVB

Not support the demo sample yet.

6.1.2. AST1030-EVB

in `samples/boards/ast1030_evb/demo`

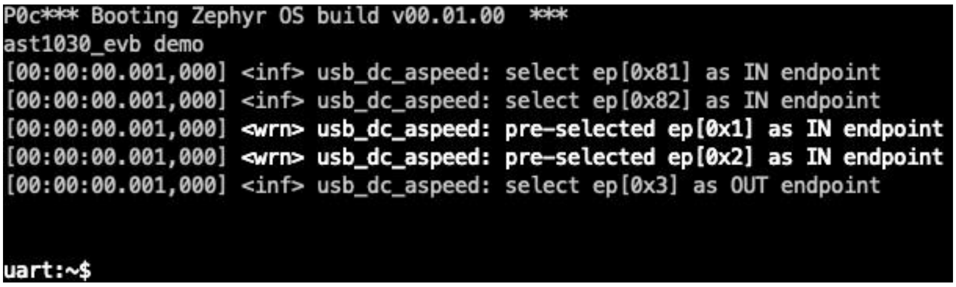
```
# tree zephyr/samples/boards/ast1030_evb/demo
samples/boards/ast1030_evb/demo
├──── CMakeLists.txt
├──── README.rst
├──── boards
│    ├──── ast1030_evb.conf
│    └──── ast1030_evb.overlay
├──── prj.conf
├──── sample.yaml
└──── src
    └──── main.c
```

To build the demo application

```
# west build -b ast1030_evb samples/boards/ast1030_evb/demo
```

The demo application uses shell command to demo the function of devices.

Boot log:



```
P0c*** Booting Zephyr OS build v00.01.00  ***
ast1030_evb demo
[00:00:00.001,000] <inf> usb_dc_aspeed: select ep[0x81] as IN endpoint
[00:00:00.001,000] <inf> usb_dc_aspeed: select ep[0x82] as IN endpoint
[00:00:00.001,000] <wrn> usb_dc_aspeed: pre-selected ep[0x1] as IN endpoint
[00:00:00.001,000] <wrn> usb_dc_aspeed: pre-selected ep[0x2] as IN endpoint
[00:00:00.001,000] <inf> usb_dc_aspeed: select ep[0x3] as OUT endpoint


uart:~$
```

Commands help:

```
Available commands:
  adc      :ADC commands
  clear    :Clear screen.
  date     :Date commands
  device   :Device commands
  gpio     :GPIO commands
  help     :Prints the help message.
  history  :Command history.
  hwinfo   :HWINFO commands
  i2c      :I2C commands
  jtag     :JTAG shell commands
  kernel   :Kernel commands
  log      :Commands for controlling logger
  md       :Mem Display command
  mw       :Mem Write command
  peci     :PECI shell commands
  pwm      :PWM shell commands
  resize   :Console gets terminal screen size or assumes default in case
the readout fails. It must be executed after each terminal width change
to ensure correct text display.
  sensor   :Sensor commands
  shell    :Useful, not Unix-like shell commands.
```

More examples of the driver API usage can be found in:

```
zephyr/samples/drivers
zephyr/tests/drivers
```

And see the driver API header file in the zephyr project:

```
zephyr/include/drivers
```

6.2. GPIO/SGPIO Driver

AST2600: Integrates two sets of Parallel GPIO Controller one set is 3.3v with maximum 208 control pins, one set is 1.8v with maximum 36 control pins, to provide general-purpose input/output functions. Two sets of Serial GPIO master, One is up to 128 SGPIO input ports and 128 output ports concurrently and Second one is up to 80. And two set of Serial GPIO slave monitors which follows SFF-8485.

AST1030: Integrates one set of Parallel GPIO Controller with maximum 151 control pins, which are 21 groups (A~U, exclude pin: M6 M7 Q5 Q6 Q7 R0 R1 R4 R5 R6 R7 S0 S3 S4 S5 S6 S7) and the group T and U are input only. One set of Serial GPIO master is up to 128 SGPIO input ports and 128 output ports concurrently.

6.2.1. Driver

6.2.1.1. Source files and configurations

interface	`include/drivers/gpio.h`
Kconfig source files	`drivers/gpio/Kconfig.aspeed`

GPIO:

source files	`drivers/gpio/gpio_aspeed.c`

Configurations	Descriptions
`CONFIG_GPIO_ASPEED`	Enable the GPIO driver support

SGPIOM:

source files	`drivers/gpio/gpio_aspeed_sgpiom.c`

Configurations	Descriptions
`CONFIG_GPIO_ASPEED_SGPIOM`	Enable the SGPIO Master driver support

6.2.2. DTS

6.2.2.1. DTS property

yaml files	`dts/bindings/gpio/aspeed,gpio.yaml`

	dts/bindings/gpio/aspeed,sgpiom.yaml

6.2.3. Usage

6.2.3.1. Shell command

source files	drivers/gpio/gpio_shell.c

Configurations	Descriptions
CONFIG_GPIO_SHELL	Enable GPIO Shell for testing

- Show device name:

```
uart:~$ device list
devices:
…
- GPIO0_U_V (READY)
- GPIO0_Q_T (READY)
- GPIO0_M_P (READY)
- GPIO0_I_L (READY)
- GPIO0_E_H (READY)
- GPIO0_A_D (READY)
```

- Usage (set direction/get value/set value):

```
uart:~$ gpio
gpio - GPIO commands
Subcommands:
  conf    :Configure GPIO: <device> <pin> <in|out|deb>
  get     :Get GPIO value: <device> <pin>
  set     :Set GPIO: <device> <pin> <0|1>
  listen  :Listen GPIO: <device> <pin> <levelH|levelL|edgeH|edgeL|edgeB>
# Set D6 to input
uart:~$ gpio conf GPIO0_A_D 30 in
Configuring GPIO0_A_D pin 30
# Set D7 to output
uart:~$ gpio conf GPIO0_A_D 31 out
Configuring GPIO0_A_D pin 31
# Set D7=0
uart:~$ gpio set GPIO0_A_D 31 0
Writing to GPIO0_A_D pin 31
```

```
uart:~$ gpio get GPIO0_A_D 30
Reading GPIO0_A_D pin 30
Value 0
# Set D7=1
uart:~$ gpio set GPIO0_A_D 31 1
Writing to GPIO0_A_D pin 31
uart:~$ gpio get GPIO0_A_D 30
Reading GPIO0_A_D pin 30
Value 1
# Set D7=0
uart:~$ gpio set GPIO0_A_D 31 0
Writing to GPIO0_A_D pin 31
uart:~$ gpio get GPIO0_A_D 30
Reading GPIO0_A_D pin 30
Value 0
# Listen D6 both edge interrupt
uart:~$ gpio conf GPIO0_A_D 30 in
Configuring GPIO0_A_D pin 30
uart:~$ gpio listen GPIO0_A_D 30 edgeB
Listen to GPIO0_A_D pin 30 mode edgeB
uart:~$ gpio set GPIO0_A_D 31 1
[00:06:14.903,000] <inf> gpio_shell: event_print: GPIO0_A_D pin 30
```

6.3. ADC Driver

ADC Engine has 2 Analog-to-Digital Convertor. Each of both has 8 voltage sensing channels. One of the 8 channels is also for battery sensing. It has internal dividing circuit. Each channel has upper and lower threshold. Larger or smaller than threshold triggers interrupt. There are second set threshold for hysteresis. Build-in a compensating method.

6.3.1. Driver

6.3.1.1. Source files and configurations

interface	`include/drivers/adc.h`
source files	`drivers/adc/aspeed/adc_aspeed.c`
Kconfig source files	`drivers/adc/Kconfig.aspeed`

Configurations	Descriptions
`CONFIG_ADC_ASPEED`	Enable the ADC driver support

6.3.2. DTS

6.3.2.1. DTS property

yaml files	`dts/bindings/adc/aspeed,adc.yaml`

6.3.2.2. Enable ADC device and set pinmux.

```
&adc0 {
    status = "okay";
# Select the channel you need.
    pinctrl-0 = <&pinctrl_adc0_default &pinctrl_adc1_default
        &pinctrl_adc2_default &pinctrl_adc3_default
        &pinctrl_adc4_default &pinctrl_adc5_default
        &pinctrl_adc6_default &pinctrl_adc7_default>;
};

&adc1 {
    status = "okay";
# Select the channel you need.
    pinctrl-0 = <&pinctrl_adc8_default &pinctrl_adc9_default
        &pinctrl_adc10_default &pinctrl_adc11_default
        &pinctrl_adc12_default &pinctrl_adc13_default
        &pinctrl_adc14_default &pinctrl_adc15_default>;
};
```

6.3.3. Usage

6.3.3.1. Shell command

source files	drivers/adc/adc_shell.c

Configurations	Descriptions
CONFIG_ADC_SHELL	Enable ADC Shell for testing

- Show device name:

```
uart:~$ device list
devices:
…
- ADC1 (READY)
  requires: PINMUX
  requires: SYSRST
  requires: ADC_CLK
- ADC0 (READY)
  requires: PINMUX
  requires: SYSRST
  requires: ADC_CLK
…
```

- Usage:

```
uart:~$ adc
adc - ADC commands
Subcommands:
  ADC0  :Select subcommand for ADC property label.

  ADC1  :Select subcommand for ADC property label.
uart:~$ adc ADC0
ADC0 - Select subcommand for ADC property label.
```

```
Subcommands:
  acq_time     :Not support
  channel      :Configure ADC channel
  gain         :Configure gain.
  print        :Print current configuration
  read         :Read adc value
               Usage: read <channel>
  reference    :Not support
  resolution   :Configure resolution
               Usage: resolution <resolution>
  calibrate    :Configure calibrate
               Usage: calibrate <1/0>
  read_format  :Configure read format
               Usage: read_format <0:raw, 1:mv>
# Configure ADC resolution to 10 bits
uart:~$ adc ADC0 resolution 10
# Configure ADC to calibration before read ADC value
uart:~$ adc ADC0 calibrate 1
# Read ADC0 channel 0
uart:~$ adc ADC0 read 0
read: 777raw
# Configure ADC print format to mv
uart:~$ adc ADC0 read_format 1
uart:~$ adc ADC0 read 0
read: 1899mv
```

6.4. PECI Driver

PECI Controller (PECI) supports PECI 1.1, 2.0, 3.0 and 4.0 protocols.

6.4.1. Driver

6.4.1.1. Source files and configurations

interface	include/drivers/peci.h
source files	drivers/peci/peci_aspeed.c
Kconfig source files	drivers/peci/Kconfig.aspeed

Configurations	Descriptions
CONFIG_PECI_ASPEED	Enable the ASPEED PECI IO driver.

6.4.2. DTS

6.4.2.1. DTS property

yaml files	dts/bindings/peci/aspeed,peci.yaml

6.4.3. Usage

6.4.3.1. Shell command

source files	drivers/peci/peci_shell.c

Configurations	Descriptions
CONFIG_PECI_SHELL	Enable PECI Shell for testing

- Show device name:

```
uart:~$ device list
devices:
…
- PECI (READY)
  requires: SYSCLK
  requires: SYSRST
…
```

- Usage:

```
uart:~$ peci
peci - PECI shell commands
Subcommands:
  init     :<device> <kbps>
  ping     :<device> <addr>
  getdib   :<device> <addr>
  gettemp  :<device> <addr>
  raw      :<device> <addr> <wr_len> <rd_len> <command(hex)> <...>
uart:~$ peci init PECI 1000
uart:~$ peci ping PECI 48
Success
uart:~$ peci getdib PECI 48
04 30 00 00 00 00 00 00
uart:~$ peci gettemp PECI 48
00 00
# Wrpkgconfig
uart:~$ peci raw PECI 48 7 1 a5 0 1 0 0 0
40
```

6.4.3.2. PECI sample app

Sample folder	samples/drivers/peci/

- Build command:

```
West build -b ast1030_evb samples/drivers/peci/
```

- Sample output:

```
PECI sample test
40
65
7a
c4
3f

peci_resp 40
Maximum temperature: 196
Start thread...
R FCS 0
```

```
Temp bytes: 0000
Temperature 32452 C
R FCS 0
Temp bytes: 0000
Temperature 32452 C
R FCS 0
```

6.5. JTAG master Driver

AST1030 support 2 identical JTAG Master controllers. JTAG Master follows Test Access Port(TAP) and state diagram IEEE 1149-1.

6.5.1. Driver

6.5.1.1. Source files and configurations

interface	`include/drivers/jtag.h`
source files	`drivers/jtag/jtag_aspeed.c`
Kconfig source files	`drivers/jtag/Kconfig.aspeed`

Configurations	Descriptions
`CONFIG_JTAG_ASPEED`	Enable the JTAG driver

6.5.2. DTS

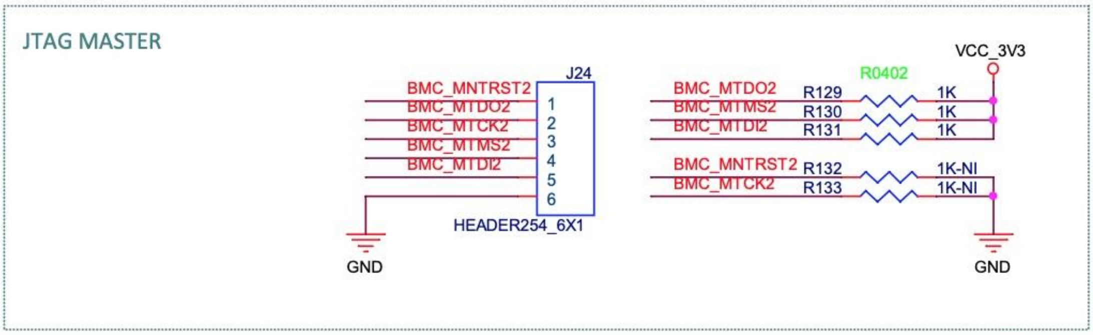
6.5.2.1. DTS property

yaml files	`dts/bindings/jtag/aspeed,jtag.yaml`

6.5.3. Usage

6.5.3.1. Board setup

JTAG pin:

- 

6.5.3.2. Shell command

source files	`drivers/jtag/jtag_shell.c`

Configurations	Descriptions
CONFIG_JTAG_SHELL	Enable JTAG Shell for testing

- Show device name:

```
uart:~$ device list
devices:
…
- JTAG1 (READY)
  requires: SYSCLK
  requires: PINMUX
  requires: SYSRST
…
```

- Usage: (Get lattice LCMXO2-7000HE device ID: 0x12b5043)

```
uart:~$ jtag
jtag - JTAG shell commands
Subcommands:
  frequency  :<device> <frequency>
  ir_scan    :<device> <len> <value>
  dr_scan    :<device> <len> <value>
  tap_set    :<device> <tap_state>
  sw_xfer    :<device> <pin> <value>
# Set jtag TCK frequency to 1MHz
uart:~$ jtag frequency JTAG1 1000000
1000000
# Set device tap state to RESET
uart:~$ jtag tap_set JTAG1 15
uart:~$ jtag ir_scan JTAG1 8 e0
5
uart:~$ jtag dr_scan JTAG1 32 0
1
2b
50
43
```

6.6. PWM/TACH Driver

PWM and Fan Tachometer Controller

- Support 16 PWM outputs and 16 fan tachometer inputs
- Support PWM frequency range from 780KHz to 24Hz
- Duty cycle from 0 to 100% with 1/256 resolution incremental
- Support fan tachometer frequency range from 1 RPM to 20K (180K) RPM
- Shared with GPIO pins

6.6.1. PWM Driver

6.6.1.1. Source files and configurations

interface	`include/drivers/pwm.h`
source files	`drivers/pwm/aspeed/pwm_aspeed.c`
Kconfig source files	`drivers/pwm/Kconfig.aspeed`

Configurations	Descriptions
`CONFIG_PWM_ASPEED`	Enable the PWM driver

6.6.2. DTS

6.6.2.1. Enable PWM device and set pinmux.

```
&pwm {
    status = "okay";
# Select the channel you need.
    pinctrl-0 = <&pinctrl_pwm0_default &pinctrl_pwm1_default
            &pinctrl_pwm2_default &pinctrl_pwm3_default
            &pinctrl_pwm4_default &pinctrl_pwm5_default
            &pinctrl_pwm6_default &pinctrl_pwm7_default
            &pinctrl_pwm8_default &pinctrl_pwm9_default
            &pinctrl_pwm10_default &pinctrl_pwm11_default
            &pinctrl_pwm12_default &pinctrl_pwm13_default
            &pinctrl_pwm14_default &pinctrl_pwm15_default>;
};
```

6.6.2.2. DTS property

yaml files	`dts/bindings/pwm/aspeed,pwm.yaml`

6.6.3. Usage

6.6.3.1. Shell command

source files	`drivers/pwm/pwm_shell.c`

Configurations	Descriptions
`CONFIG_PWM_SHELL`	Enable the PWM related shell commands.

- Show device name:

```
uart:~$ device list
devices:
…
- PWM (READY)
  requires: PINMUX
  requires: SYSRST
  requires: SYSCLK
…
```

- Usage:

```
uart:~$ pwm
pwm - PWM shell commands
Subcommands:
  cycles  :<device> <pwm> <period in cycles> <pulse width in cycles> [flags]
  usec    :<device> <pwm> <period in usec> <pulse width in usec> [flags]
  nsec    :<device> <pwm> <period in nsec> <pulse width in nsec> [flags]
# Set pwm channel 0 frequency and duty cycle to 50%
uart:~$ pwm nsec PWM 0 40000 20000
```

6.6.4. Tach Driver

6.6.4.1. Source files and configurations

interface	`include/drivers/sensor.h`
source files	`drivers/sensor/aspeed/tach_aspeed.c`
Kconfig source files	`drivers/sensor/aspeed/Kconfig`

Configurations	Descriptions
CONFIG_TACH_ASPEED	Enable the TACH driver

6.6.5. DTS

6.6.5.1. Enable PWM device and set pinmux.

```
&tach {
    status = "okay";
# Select the channel you need.

    pinctrl-0 = <&pinctrl_tach0_default &pinctrl_tach1_default
            &pinctrl_tach2_default &pinctrl_tach3_default
            &pinctrl_tach4_default &pinctrl_tach5_default
            &pinctrl_tach6_default &pinctrl_tach7_default
            &pinctrl_tach8_default &pinctrl_tach9_default
            &pinctrl_tach10_default &pinctrl_tach11_default
            &pinctrl_tach12_default &pinctrl_tach13_default
            &pinctrl_tach14_default &pinctrl_tach15_default>;
# Create fan device to monitor.
    fan@0 {
        reg = <0x0>;
        pulse-pr = <2>;
        min-rpm = <1000>;
        tach-div = <5>;
        label = "FAN0";
    };
};
```

6.6.5.2. DTS property

yaml files	dts/bindings/tach/aspeed,tach.yaml

The dividing value and RPM range are listed below. Chose 6 to be compatible with most of the fan.

Tach divide	Max rpm	Min rpm
0	6000000000	5722
1	1500000000	1430
2	375000000	357

3	93750000	89
4	23437500	22
5	5859375	5
6	1464843	1
7	366210	0
8	91552	0
9	22888	0
10	5722	0
11	1430	0

6.6.6. Usage

6.6.6.1. Shell command

source files	drivers/sensor/sensor_shell.c

Configurations	Descriptions
CONFIG_ SENSOR_SHELL	This shell provides access to basic sensor data.

- Show device name:

```
uart:~$ device list
devices:
…
- FAN0 (READY)
  requires: PINMUX
  requires: SYSRST
  requires: SYSCLK
…
```

- Usage:

```
uart:~$ sensor
sensor - Sensor commands
Subcommands:
  get  :Get sensor data. Channel names are optional. All channels are read
when
        no channels are provided. Syntax:
        <device_name> <channel name 0> .. <channel name N>
uart:~$ sensor get FAN0
```

```
channel idx=37 rpm = 1692.000000
```

6.7. eSPI Driver

Enhanced Serial Peripheral Interface (eSPI) is an interface using pins of SPI, but runs different protocol. The interface supports peripheral, virtual wire, out-of-band, and flash sharing channels. This controller supports all 4 channels and operates at max frequency of 66MHz.

6.7.1. Driver

6.7.1.1. Source files and configurations

interface	`include/drivers/espi.h`
source files	`drivers/espi/espi_aspeed.c`
Kconfig source files	`drivers/espi/Kconfig.aspeed`

Configurations	Descriptions
`CONFIG_ESPI_ASPEED`	Enable ESPI driver support

6.7.2. DTS

6.7.2.1. Enable eSPI device

```
&espi {
     status = "okay";
     perif,memcyc-src-addr = <0x98000000>;
     perif,memcyc-size = <0x10000>;
     flash,safs-mode = <0x1>;
};
```

6.7.2.2. DTS property

yaml files	`dts/bindings/espi/aspeed,espi.yaml`

6.7.3. Usage

sample app	`samples/drivers/espi_aspeed/oob/src/main.c` `samples/drivers/espi_aspeed/safs/src/main.c`

6.8. I2C Driver

The I2C driver supports the communication between Master and Slave.

6.8.1. Driver

6.8.1.1. Source files and configurations

interface	include/drivers/i2c.h
source files	drivers/i2c/i2c_aspeed.c
Kconfig source files	Drivers/i2c/Kconfig.aspeed

Configurations	Descriptions
CONFIG_DEVICE_I2C	I2C device and driver Device Drivers-> i2c drivers-> [*] Aspeed I2C driver

6.8.1.2. Device properties

struct i2c_aspeed_config	
uintptr_t base;	I2c device base address.
void (*irq_config_func)(...);	I2c device interrupt function.
uint32_t bitrate;	I2c bus clock bit rate.
const struct device *clock_dev;	Pointer to the device structure for the clock controller driver instance.
const clock_control_subsys_t clk_id;	Pointer to an opaque data representing the sub-system.

6.8.2. Virtual Slave Driver

The I2C virtual slave drivers are designed as virtual I2C slave device. When this device is attached into any one I2C device that is set as an I2C slave.

6.8.2.1. I2c EEPROM Virtual Slave Driver

6.8.2.1.1. Source files and configurations

This driver is designed as a virtual I2C EEPROM slave device that is contained 256-byte size buffer.

interface	include/drivers/i2c/slave/eeprom_slave.h

source files	drivers/i2c/slave/eeprom_slave.c
Kconfig source files	Drivers/i2c/slave/Kconfig.eeprom

Configurations	Descriptions
CONFIG_I2C_EEPROM_SLAVE	I2C EEPROM virtual slave device Device Drivers-> i2c drivers-> [*] i2c slave drivers-> [*] I2C slave EEPROM driver

6.8.2.1.1. Device properties

struct i2c_eeprom_slave_config	
char *controller_dev_name;	Virtual EEPROM device name.
uint8_t address;	Virtual EEPROM I2C slave address.
uint32_t buffer_size;	Virtual EEPROM size.
uint8_t *buffer;	Pointer to the virtual EEPROM buffer.

6.8.2.2. I2c IPMB Virtual Slave Driver

6.8.2.2.1. Source files and configurations

This driver is designed as a virtual I2C IPMB buffer slave device that is contained user defined size buffer.

interface	include/drivers/i2c/slave/ipmb.h
source files	drivers/i2c/slave/ipmb_slave.c
Kconfig source files	drivers/i2c/slave/Kconfig.ipmb

Configurations	Descriptions
CONFIG_I2C_IPMB_SLAVE	I2C IPMB virtual slave device Device Drivers-> i2c drivers-> [*] i2c slave drivers-> [*] I2C slave IPMB driver

6.8.2.2.2. Device properties

struct i2c_eeprom_slave_config	
char *controller_dev_name;	Virtual IPMB device name.
uint8_t address;	Virtual IPMB I2C slave address.
uint32_t ipmb_msg_length;	Virtual IPMB buffer size.

6.8.3. Usage

6.8.3.1. Source files and configurations

source files	`drivers/i2c/i2c_shell.c`
Kconfig source files	`drivers/i2c/Kconfig`

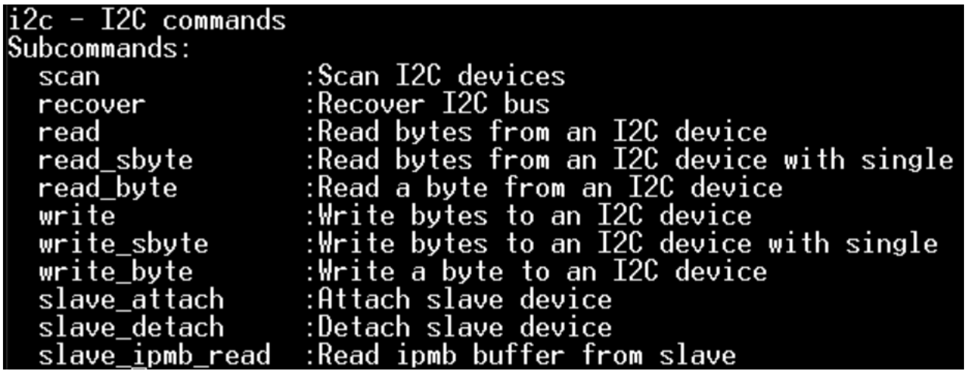
Configurations	Descriptions
`CONFIG_I2C_SHELL`	Enable I2C Shell. `Device Drivers-> i2c drivers-> i2c drivers-> [*] Enable I2C Shell`

6.8.3.2. Board setup

```
/* declare I2C devices */
#if CONFIG_DEVICE_I2C
DECLARE_DEV_CLK(i2c_global, 0, 0, 0);
DECLARE_DEV_RESET(i2c_global, SCU_BASE + 0x50, SCU_BASE + 0x54, BIT(2));
DECLARE_DEV(i2c_global, ASPEED_DEV_I2C_GLOBAL, I2C_GLOBAL_BASE, NULL);
```

6.8.3.3. Usage and examples

-I2C master command list

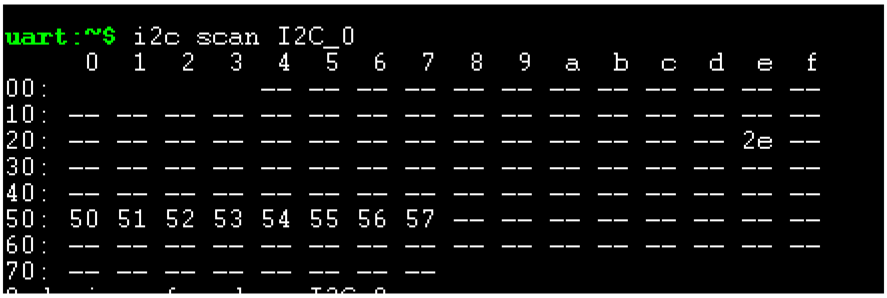


```
i2c - I2C commands
Subcommands:
  scan              :Scan I2C devices
  recover           :Recover I2C bus
  read              :Read bytes from an I2C device
  read_sbyte        :Read bytes from an I2C device with single
  read_byte         :Read a byte from an I2C device
  write             :Write bytes to an I2C device
  write_sbyte       :Write bytes to an I2C device with single
  write_byte        :Write a byte to an I2C device
  slave_attach      :Attach slave device
  slave_detach      :Detach slave device
  slave_ipmb_read   :Read ipmb buffer from slave
```

AST1030 EVB Expected result:

Example : i2c scan bus#0

#i2c scan I2C_0

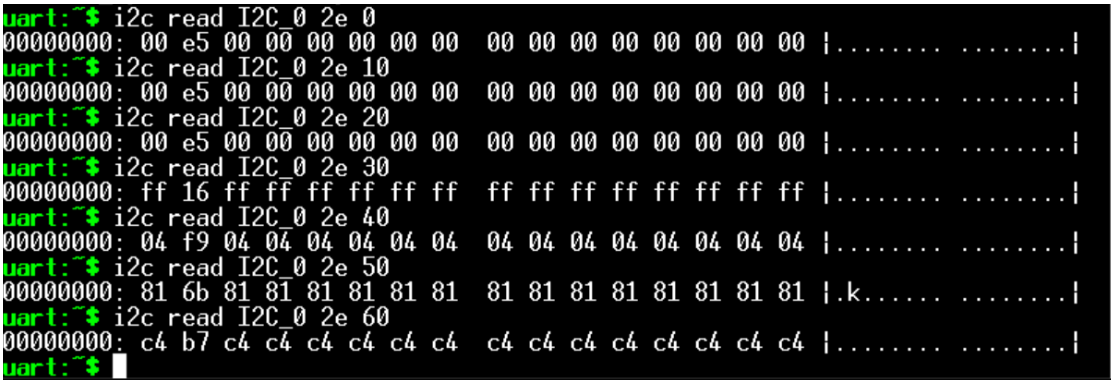


```
uart:~$ i2c scan I2C_0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:             -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- 2e --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 51 52 53 54 55 56 57 -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Example : ADT7490 default is address 0x2e

#Read offset 0x0 ~ 0x60 on 0x2e every 16 byte

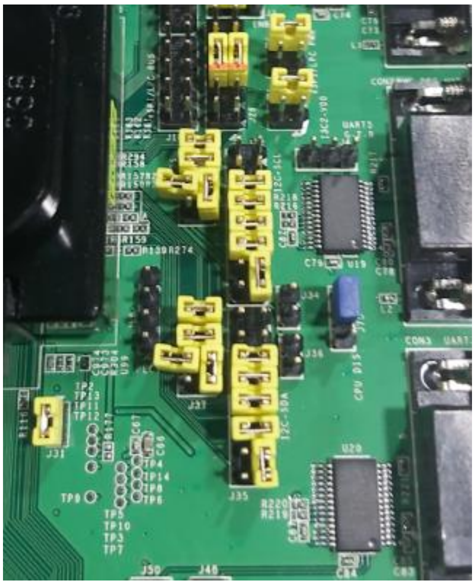
uart:~$ i2c read I2C_0 2e (0x0 ~ 0x60)



```
uart:~$ i2c read I2C_0 2e 0
00000000: 00 e5 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |........ ........|
uart:~$ i2c read I2C_0 2e 10
00000000: 00 e5 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |........ ........|
uart:~$ i2c read I2C_0 2e 20
00000000: 00 e5 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |........ ........|
uart:~$ i2c read I2C_0 2e 30
00000000: ff 16 ff ff ff ff ff ff  ff ff ff ff ff ff ff ff |........ ........|
uart:~$ i2c read I2C_0 2e 40
00000000: 04 f9 04 04 04 04 04 04  04 04 04 04 04 04 04 04 |........ ........|
uart:~$ i2c read I2C_0 2e 50
00000000: 81 6b 81 81 81 81 81 81  81 81 81 81 81 81 81 81 |.k...... ........|
uart:~$ i2c read I2C_0 2e 60
00000000: c4 b7 c4 c4 c4 c4 c4 c4  c4 c4 c4 c4 c4 c4 c4 c4 |........ ........|
uart:~$
```

-I2C slave loop setting on AST1030 EVB

Connect i2c#1 <-> i2c#2 with J35/J32 as below setting



-I2C slave command list (virtual EEPROM slave type)

Please add below setting into dts file Ex. Add a virtual EEPROM (address: 0x40) and virtual IPMB (address: 0x50) slave device into i2c bus#1

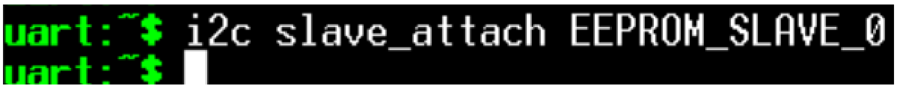
```
&i2c0 {
    status = "okay";
    pinctrl-0 = <&pinctrl_i2c0_default>;

    eeprom@40 {
        compatible = "atmel,at24";
        reg = <0x40>;
        label = "EEPROM_SLAVE_0";
        size = <256>;
        pagesize = <16>;
        address-width = <8>;
        timeout = <5>;
#ifdef CONFIG_I2C_EEPROM_SLAVE
        status = "okay";
#endif
    };

    ipmb@50 {
        compatible = "aspeed,ipmb";
        reg = <0x50>;
        label = "IPMB_SLAVE_0";
        size = <5>;
#ifdef CONFIG_I2C_IPMB_SLAVE
        status = "okay";
#endif
    };
};
```

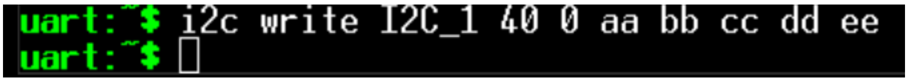
After select I2C EEPROM slave and I2C IPMB slave type virtual device in menu config.

Ex. Attach virtual EEPROM slave device into I2C_0
#i2c slave_attach EEPROM_SLAVE_0



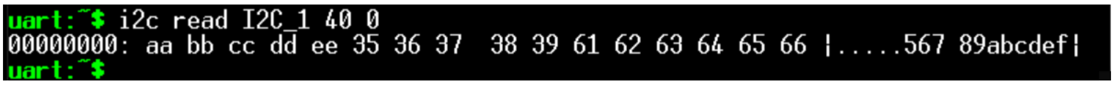
```
uart:~$ i2c slave_attach EEPROM_SLAVE_0
uart:~$
```

Ex. Write virtual EEPROM device (address is 0x40) with I2C_1 from offset 0 ~ 4 with 0xaa 0xbb 0xcc 0xdd 0xee.



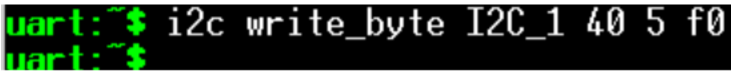
```
uart:~$ i2c write I2C_1 40 0 aa bb cc dd ee
uart:~$
```

Ex. Read virtual EEPROM device (address is 0x40) with I2C_1 from offset 0 ~ 0x10.



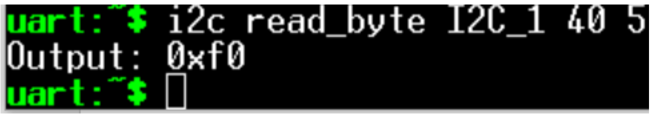
```
uart:~$ i2c read I2C_1 40 0
00000000: aa bb cc dd ee 35 36 37  38 39 61 62 63 64 65 66 |.....567 89abcdef|
uart:~$
```

Ex. Write virtual EEPROM device (address is 0x40) with I2C_1 from offset 0x5 with single byte 0xf0.



```
uart:~$ i2c write_byte I2C_1 40 5 f0
uart:~$
```

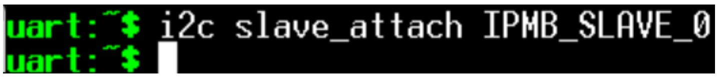
Ex. Read virtual EEPROM device (address is 0x40) with I2C_1 from offset 0x5.



```
uart:~$ i2c read_byte I2C_1 40 5
Output: 0xf0
uart:~$
```

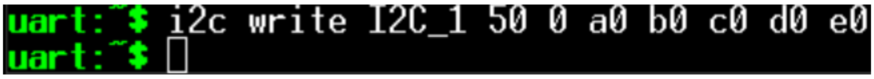
-I2C slave command list (virtual IPMB slave type)

Ex. Attach virtual IPMB slave device into I2C_0
#i2c slave_attach IPMB_SLAVE_0



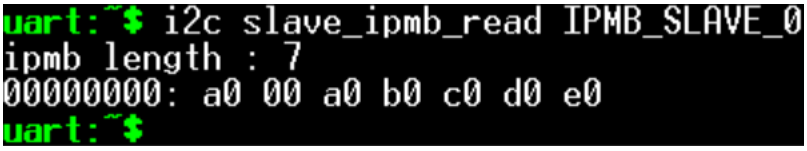
```
uart:~$ i2c slave_attach IPMB_SLAVE_0
uart:~$
```

Ex. Write virtual IPMB device (address is 0x50) with I2C_1 from offset 0 ~ 4 with 0xa0 0xb0 0xc0 0xd0 0xe0.



```
uart:~$ i2c write I2C_1 50 0 a0 b0 c0 d0 e0
uart:~$
```

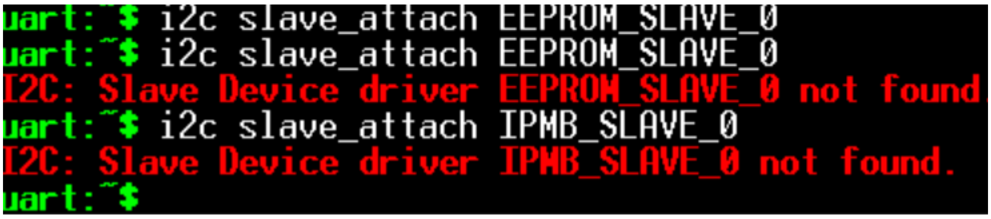
Ex. Read virtual IPMB device with I2C_1 from the buffer. This buffer is designed as FIFO.



```
uart:~$ i2c slave_ipmb_read IPMB_SLAVE_0
ipmb length : 7
00000000: a0 00 a0 b0 c0 d0 e0
uart:~$
```

-I2C slave detach command (virtual slave)

If you want to change another type virtual driver, you should detach existed attached slave device first. Or you would not be allowed to attached another one virtual slave device even the type is same.

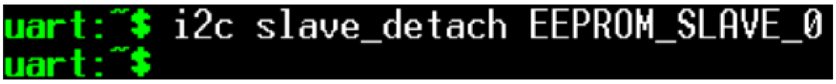


```
uart:~$ i2c slave_attach EEPROM_SLAVE_0
uart:~$ i2c slave_attach EEPROM_SLAVE_0
I2C: Slave Device driver EEPROM_SLAVE_0 not found.
uart:~$ i2c slave_attach IPMB_SLAVE_0
I2C: Slave Device driver IPMB_SLAVE_0 not found.
uart:~$
```

The correct steps are shown below:

Detach the EEPROM virtual device by

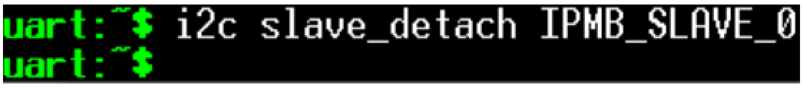
#i2c slave_detach EEPROM_SLAVE_0



```
uart:~$ i2c slave_detach EEPROM_SLAVE_0
uart:~$
```

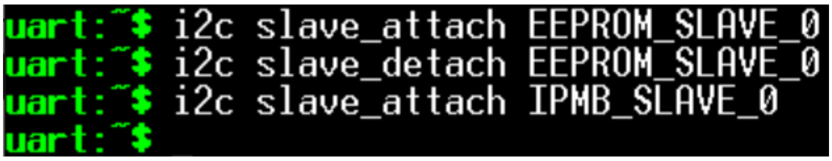
Detach the IPMB virtual device by

#i2c slave_detach IPMB_SLAVE_0



```
uart:~$ i2c slave_detach IPMB_SLAVE_0
uart:~$
```

Attach the EEPROM virtual device. Then detach it and attach IPMB virtual device.



```
uart:~$ i2c slave_attach EEPROM_SLAVE_0
uart:~$ i2c slave_detach EEPROM_SLAVE_0
uart:~$ i2c slave_attach IPMB_SLAVE_0
uart:~$
```

6.9. IPMI KCS Driver

The KCS driver supports the communication between BMC and SMS through IPMI KCS interface. There is total 4 KCS channels supported.

6.9.1. Driver

6.9.1.1. Source files and configurations

interface	`drivers/ipmi/kcs_aspeed.c`
source files	`drivers/ipmi/kcs_aspeed.c`
Kconfig source files	`drivers/Kconfig.aspeed`

Configurations	Descriptions
`CONFIG_KCS_ASPEED`	Enable KCS driver support

6.9.2. DTS

6.9.2.1. Enable KCS device

```
&kcs1 {
    status = "okay";
    addr = <0xca0>;
};
&kcs2 {
    status = "okay";
    addr = <0xca8>;
};
&kcs3 {
    status = "okay";
    addr = <0xca2>;
};
&kcs4 {
    status = "okay";
    addr = <0xca4>;
};
```

6.9.2.2. DTS property

yaml files	`dts/bindings/ipmi/aspeed,kcs.yaml`

6.9.3. Usage

sample app	`samples/drivers/ipmi/aspeed/kcs/src/main.c`

6.10. IPMI BT Driver

The BT driver supports the communication between BMC and SMS through IPMI BT interface.

6.10.1. Driver

6.10.1.1. Source files and configurations

interface	`drivers/ipmi/kcs_aspeed.c`
source files	`drivers/ipmi/kcs_aspeed.c`
Kconfig source files	`drivers/Kconfig.aspeed`

Configurations	Descriptions
`CONFIG_KCS_ASPEED`	Enable BT driver support

6.10.2. DTS

6.10.2.1. Enable BT device

```
&bt {
     status = "okay";
     addr = <0xe4>;
     sirq = <10>;
};
```

6.10.2.2. DTS property

yaml files	`dts/bindings/ipmi/aspeed,bt.yaml`

6.10.3. Usage

sample app	`samples/drivers/ipmi/aspeed/bt/src/main.c`

6.11. USB Device Driver

AST1030 USB Device Controller is for USB device communicates with USB host. The Driver is implemented as composite device which supports multiple device class.
One is Communication USB Device Interface Class Abstract Control Model interface (CDC ACM). The other is Application Specific USB Device Interface Class for Device Firmware Upgrade (DFU).

6.11.1. Driver

6.11.1.1. Source files and configurations

interface	`include/drivers/usb/usb_dc.h`
source files	`drivers/usb/device/usb_dc_aspeed.c`
Kconfig source files	`drivers/usb/device/Kconfig`

Configurations	Descriptions
`CONFIG_USB_ASPEED`	USB device and driver

6.11.2. DTS

```
&udc {
    status = "okay";
};
```

6.11.3. Usage

6.11.3.1. Shell command

source files	`drivers/usb/usb_shell.c`

Configurations	Descriptions
`CONFIG_SHELL_CMDS_USB`	Enable USB Shell commands for testing

- Usage:

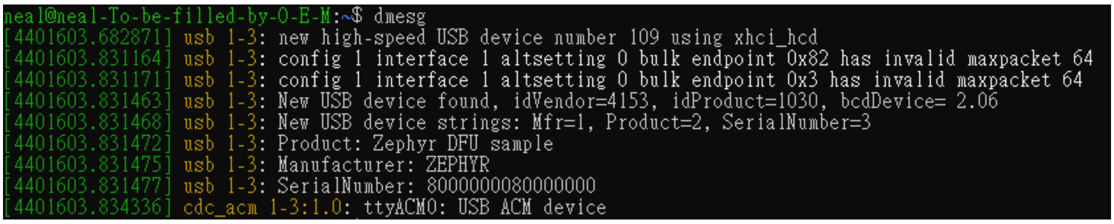
```
uart:~$ usb enable
enter cmd_usb_enable...
```

```
This device supports USB CDC_ACM class.

[00:00:07.751,000] <inf> usb_cdc_acm: Device configured
uart:~$
```

6.11.3.2. Host Side

Plug into Host side log

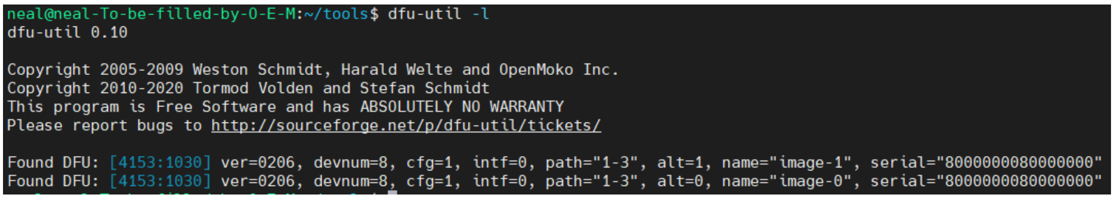


```
neal@neal-To-be-filled-by-O-E-M:~$ dmesg
[4401603.682871] usb 1-3: new high-speed USB device number 109 using xhci_hcd
[4401603.831164] usb 1-3: config 1 interface 1 altsetting 0 bulk endpoint 0x82 has invalid maxpacket 64
[4401603.831171] usb 1-3: config 1 interface 1 altsetting 0 bulk endpoint 0x3 has invalid maxpacket 64
[4401603.831463] usb 1-3: New USB device found, idVendor=4153, idProduct=1030, bcdDevice= 2.06
[4401603.831468] usb 1-3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[4401603.831472] usb 1-3: Product: Zephyr DFU sample
[4401603.831475] usb 1-3: Manufacturer: ZEPHYR
[4401603.831477] usb 1-3: SerialNumber: 8000000080000000
[4401603.834336] cdc_acm 1-3:1.0: ttyACM0: USB ACM device
```

6.11.3.3. DFU

1. Get dfu-util from host side
   - For Ubuntu, apt install to get it.
   - For AST2600, you should get source code and cross compile for arm system.
   - Check this for more details: http://dfu-util.sourceforge.net/

List currently attached DFU capable devices.



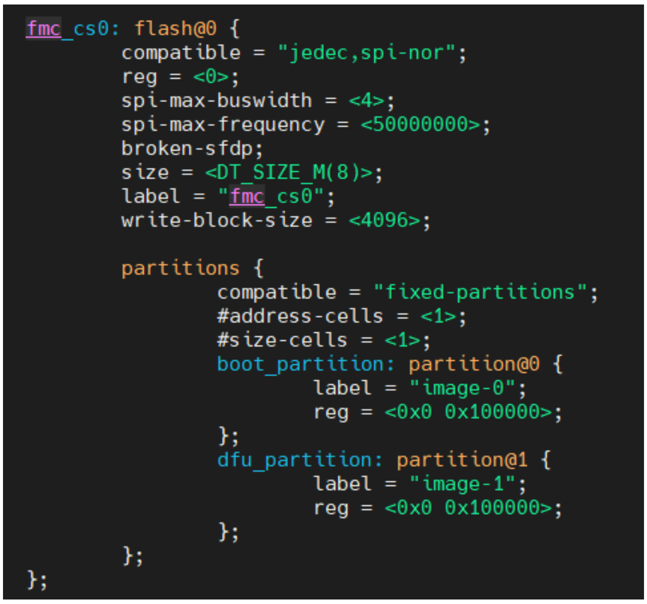
```
neal@neal-To-be-filled-by-O-E-M:~/tools$ dfu-util -l
dfu-util 0.10

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2020 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

Found DFU: [4153:1030] ver=0206, devnum=8, cfg=1, intf=0, path="1-3", alt=1, name="image-1", serial="8000000080000000"
Found DFU: [4153:1030] ver=0206, devnum=8, cfg=1, intf=0, path="1-3", alt=0, name="image-0", serial="8000000080000000"
```

You can see 2 alternates for this DFU devices, which is defined in dts. But it can only download image at label “image-1”. So you should pre-define your dfu_partition start address and range at build time.

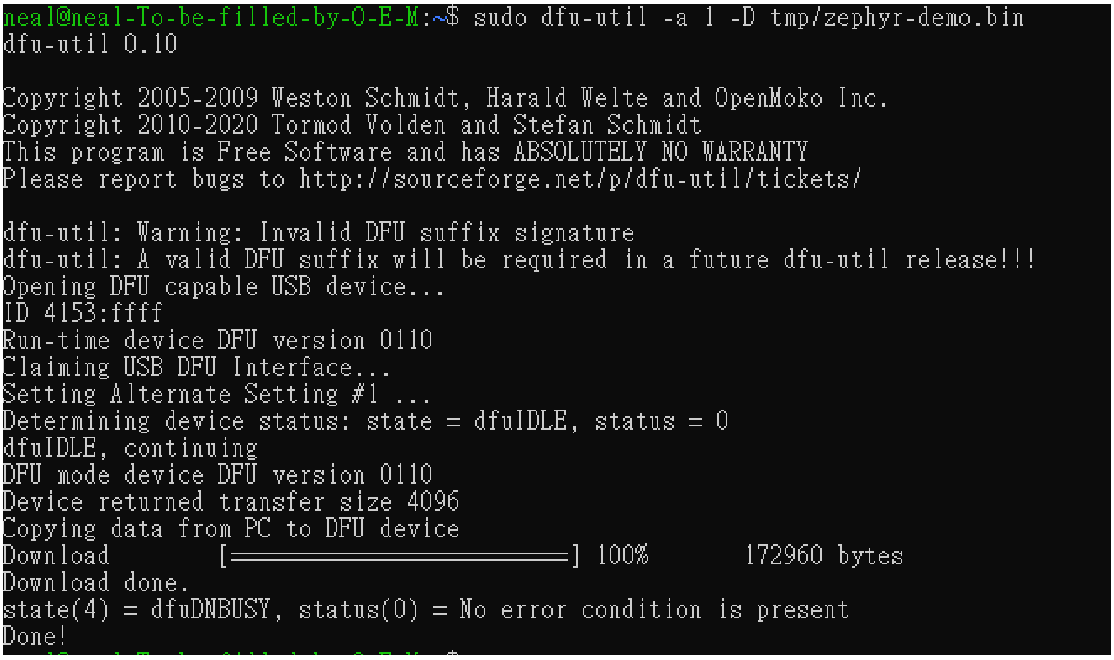
File: dts/arm/aspeed/ast10x0.dtsi



```
fmc_cs0: flash@0 {
        compatible = "jedec,spi-nor";
        reg = <0>;
        spi-max-buswidth = <4>;
        spi-max-frequency = <50000000>;
        broken-sfdp;
        size = <DT_SIZE_M(8)>;
        label = "fmc_cs0";
        write-block-size = <4096>;

        partitions {
                compatible = "fixed-partitions";
                #address-cells = <1>;
                #size-cells = <1>;
                boot_partition: partition@0 {
                        label = "image-0";
                        reg = <0x0 0x100000>;
                };
                dfu_partition: partition@1 {
                        label = "image-1";
                        reg = <0x0 0x100000>;
                };
        };
};
```

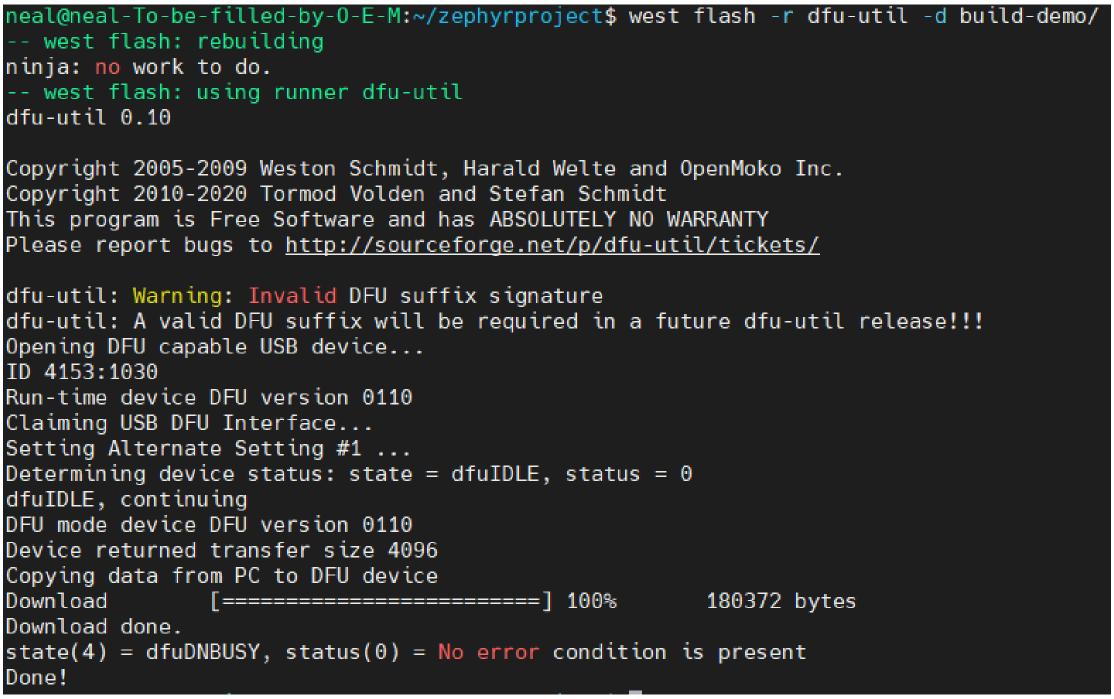
2. Download image to spi flash
    - Method 1: $ sudo dfu-util -a 1 -D {file}
    - Method 2: $ west flash -r dfu-util -d build-demo
    - After download done, you can boot from spi flash !



```
neal@neal-To-be-filled-by-O-E-M:~$ sudo dfu-util -a 1 -D tmp/zephyr-demo.bin
dfu-util 0.10

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2020 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

dfu-util: Warning: Invalid DFU suffix signature
dfu-util: A valid DFU suffix will be required in a future dfu-util release!!!
Opening DFU capable USB device...
ID 4153:ffff
Run-time device DFU version 0110
Claiming USB DFU Interface...
Setting Alternate Setting #1 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0110
Device returned transfer size 4096
Copying data from PC to DFU device
Download        [=========================] 100%       172960 bytes
Download done.
state(4) = dfuDNBUSY, status(0) = No error condition is present
Done!
```



```
neal@neal-To-be-filled-by-O-E-M:~/zephyrproject$ west flash -r dfu-util -d build-demo/
-- west flash: rebuilding
ninja: no work to do.
-- west flash: using runner dfu-util
dfu-util 0.10

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2020 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

dfu-util: Warning: Invalid DFU suffix signature
dfu-util: A valid DFU suffix will be required in a future dfu-util release!!!
Opening DFU capable USB device...
ID 4153:1030
Run-time device DFU version 0110
Claiming USB DFU Interface...
Setting Alternate Setting #1 ...
Determining device status: state = dfuIDLE, status = 0
dfuIDLE, continuing
DFU mode device DFU version 0110
Device returned transfer size 4096
Copying data from PC to DFU device
Download        [=========================] 100%       180372 bytes
Download done.
state(4) = dfuDNBUSY, status(0) = No error condition is present
Done!
```

6.12. UART Driver

UART (Universal Asynchronous Receiver/Transmitter) providing serial communication capabilities with other external devices, like another computer using a serial cable based on RS232 protocol. This core is designed to be compatible with the industry standard - 16550 UART. The integrated UART are listed as the following tables. Certain UART can be leveraged as system UART for Host use through LPC/eSPI. Each UART can support up to 115200 baud rate and possesses transmit and receive FIFO buffers to reduce CPU interrupts.

In addition, the UART DMA (UDMA) is also supported to further offload the CPU overhead. Note that each UDMA channel is dedicated to a specific UART device as show in the following table.

UART device	UDMA channel number
UART1	0
UART2	1
UART3	2
UART4	3
UART5	N/A
UART6	4
UART7	5
UART8	6
UART9	7
UART10	8
UART11	9
UART12	10
UART13	11

6.12.1. Driver

6.12.1.1. Source files and configurations

interface	`include/drivers/uart.h`
source files	`drivers/serial/uart_aspeed`
Kconfig source files	`drivers/serial/Kconfig.aspeed`

Configurations	Descriptions
`CONFIG_UART_ASPEED`	Enable UART driver support

6.12.2. DTS

6.12.2.1. Enable UART device

```
// UART w/o DMA
&uart3 {
    status = "okay";
};

// UART w/ DMA
&uart4 {
    status = "okay";
    dma = <1>;
    dma,channel = <3>;
};
```

6.12.2.2. DTS property

yaml files	`dts/bindings/serial/aspeed,uart.yaml`

6.13. Virtual UART Driver

Virtual UART provides virtual serial communication capabilities between the Host and the BMC. The VUART is equipped with two sets of registers compatible with the industry defector standard - 16550 UART. One set is for host CPU whereas the other set is for ARM CPU. Host CPU and ARM CPU can communicate with each other like there is a physical UART link between them, but the related data transfer actually is just through pure register read and write transfers in the chip. The base address for host CPU to access UART registers through LPC/eSPI can be programmed by ARM CPU by the extended related registers (VxUART28 and VxUART2C).

VUART shares the same driver with physical UART with additional DTS properties, namely “virtual”, “virtual,port”, “virtual,sirq”, and “virtual,sirq-polarity”, are required. The table gives a reference for the configuration of the VUART dedicated DTS properties.

VUART Device	Port	SIRQ	SIRQ Polarity (LPC)	SIRQ Polarity (eSPI)	SIRQ Polarity (PCIe)
VUART1	0x3F8	0x4	1	0	N/A
VUART2	0x3F8	0x3	1	0	N/A
VUART3	0x3E8	0x4	N/A	N/A	0
VUART4	0x3E8	0x3	N/A	N/A	0

Similar to physical UART devices, the UDMA can be applied on VUART and the corresponding DMA channel numbers are listed as the following table.

Virtual UART device	UDMA channel number
VUART1	12
VUART2	13

6.13.1. Driver

6.13.1.1. Source files and configurations

interface	`include/drivers/uart.h`
source files	`drivers/serial/uart_aspeed`
Kconfig source files	`drivers/serial/Kconfig.aspeed`

Configurations	Descriptions

CONFIG_UART_ASPEED	Enable UART driver support

6.13.2. DTS

6.13.2.1. Enable UART device

```
// VUART w/o DMA
&vuart1 {
    status = "okay";
    virtual,port = <0x3f8>;
    virtual,sirq = <4>;
    virtual,sirq-polarity = <0>;
};

// VUART w/ DMA
&vuart2 {
    status = "okay";
    virtual,port = <0x3f8>;
    virtual,sirq = <4>;
    virtual,sirq-polarity = <0>;
    dma = <1>;
    dma,channel = <13>;
};
```

6.13.2.2. DTS property

yaml files	dts/bindings/serial/aspeed,uart.yaml

6.14. Snoop Driver

This driver snoops the data bytes written to LPC I/O ports by the Host. The device supports up to 2 I/O ports, simultaneous snooping.

6.14.1. Driver

6.14.1.1. Source files and configurations

interface	`include/driver/misc/aspeed/snoop_aspeed.h`
source files	`drivers/misc/aspeed/snoop_aspeed.c`
Kconfig source files	`drivers/misc/aspeed/Kconfig`

Configurations	Descriptions
`CONFIG_SNOOP_ASPEED`	Enable Snoop driver support

6.14.2. DTS

6.14.2.1. Enable snoop device

```
&snoop {
     status = "okay";
     port = <0x80>, <0x81>;
};
```

6.14.2.2. DTS property

yaml files	`dts/bindings/misc/aspeed,snoop.yaml`

6.14.3. Usage

sample app	`samples/drivers/misc/aspeed/snoop/src/main.c`

6.15. PCC Driver

This driver snoops the data bytes written to LPC I/O ports by the Host. This device supports flexible port range to snoop.

6.15.1. Driver

6.15.1.1. Source files and configurations

interface	`include/driver/misc/aspeed/pcc_aspeed.h`
source files	`drivers/misc/aspeed/pcc_aspeed.c`
Kconfig source files	`drivers/misc/aspeed/Kconfig`

Configurations	Descriptions
`CONFIG_PCC_ASPEED`	Enable Snoop driver support

6.15.2. DTS

The DTS properties are named in accordance with the HW bit fields. For the complete explanation, please refer to the description of PCCRx registers in the datasheet.

6.15.2.1. Enable PCC device to snoop port 80h

```
&pcc {
    status = "okay";
    addr = <0x80>;
    addr-xbit = <0x0>;
    addr-hbit-sel = <0x1>;
    rec-mode = <0x1>;
    dma-mode;
};
```

6.15.2.2. Enable PCC device to snoop ports 80h – 81h

```
&pcc {
    status = "okay";
    addr = <0x80>;
    addr-xbit = <0x1>;
    addr-hbit-sel = <0x1>;
    rec-mode = <0x1>;
    dma-mode;
};
```

6.15.2.3. Enable PCC device to snoop ports 80h – 83h

```
&pcc {
    status = "okay";
    addr = <0x80>;
    addr-xbit = <0x3>;
    addr-hbit-sel = <0x1>;
    rec-mode = <0x1>;
    dma-mode;
};
```

6.15.2.4. DTS property

yaml files	`dts/bindings/misc/aspeed,pcc.yaml`

6.15.3. Usage

The PCC driver delivers the raw data directly read from the HW. For the data output format, please refer to the datasheet.

sample app	`samples/drivers/misc/aspeed/pcc/src/main.c`

6.15.4. Limitation with eSPI

When PCC is used to snoop debug code over eSPI, the underlying HW response to the eSPI master is NON_FATAL_ERROR instead of ACCEPT. And the reaction of eSPI master on receiving NFE is unpredictable. It depends on the Host side FW implementation. Therefore, it is not recommended to use PCC over eSPI.

If PCC over eSPI is a must, the following SW workaround can be leveraged to prevent the NFE response. Note that the SW workaround has certain limitations and is mutually exclusive to the Snoop driver.

- Port range is limited to a continuous, 4-bytes region
- Disable Snoop driver
- Assume PCC target port is X, set SNPWADR[31:16]=X+2, SNPWADR[15:0]=X
- Set HICRB[15:14]=11b
- Set HICR6[19]=1b

6.16. Mailbox Driver

The mailbox driver supports 32 general purpose registers for the data communication between the Host and the BMC.

6.16.1. Driver

6.16.1.1. Source files and configurations

interface	`include/driver/misc/aspeed/mbox_aspeed.h`
source files	`drivers/misc/aspeed/mbox_aspeed.c`
Kconfig source files	`drivers/misc/aspeed/Kconfig`

Configurations	Descriptions
`CONFIG_MAILBOX_ASPEED`	Enable Mailbox driver support

6.16.2. DTS

6.16.2.1. Enable mailbox device

```
&mbox {
    status = "okay";
};
```

6.16.2.2. DTS property

yaml files	`dts/bindings/misc/aspeed,mbox.yaml`

6.16.3. Usage

sample app	`samples/drivers/misc/aspeed/mailbox/src/main.c`

6.17. FMC_SPI Driver

SPI flash controller driver

- Support two chip selects for each SPI controller.
- Support 1-1-1, 1-1-2, 1-2-2 and 1-1-4 SPI mode or protocol.
- Support SFDP basic and 4-byte address instruction parameters parser.
- Support timing calibration for 100MHz SPI clock frequency.

6.17.1. Driver

6.17.1.1. Source files and configurations

interface	`include/drivers/flash.h`
source files	`drivers/spi/spi_aspeed.c` `drivers/flash/spi_nor_multi_dev.c`
Kconfig source files	`drivers/spi/Kconfig.aspeed` `drivers/flash/Kconfig.multi_dev`

Configurations	Descriptions

`CONFIG_SPI_ASPEED=y`
`CONFIG_FLASH=y`
`CONFIG_SPI_NOR_MULTI_DEV=y`
`CONFIG_FLASH_SHELL=y`
`CONFIG_HEAP_MEM_POOL_SIZE=16384`
	- Enable ASPEED SPI controller driver.
	- Enable SPI NOR flash driver for multiple chip select.

6.17.2. DTS

6.17.2.1. Enable SPI flash device.

```
        fmc: spi@7e620000 {
            compatible = "aspeed,spi-controller";
            reg = <0x7e620000 0xc4>, <0x80000000 0x10000000>;
            reg-names = "ctrl_reg", "spi_mmap";
            clocks = <&sysclk ASPEED_CLK_HCLK>;
            pinctrl-0 = <&pinctrl_fmc_quad>;
            num-cs = <2>;
            ast-platform = <1030>;
            ctrl-type = "bspi";
            label = "FMC";
            #address-cells = <1>;
            #size-cells = <0>;
            spi-ctrl-caps-mask = <0x000e0c0c>;
            status = "disabled";

            flash@0 {
                compatible = "jedec,spi-nor";
                reg = <0>;
                spi-max-buswidth = <1>;
                spi-max-frequency = <25000000>;
                broken-sfdp;
                size = <DT_SIZE_M(8)>;
                label = "fmc_cs0";
                status = "disabled";
            };

            flash@1 {
                compatible = "jedec,spi-nor";
                reg = <1>;
                spi-max-buswidth = <1>;
                spi-max-frequency = <25000000>;
                label = "fmc_cs1";
                status = "disabled";
            };
        };

        spi1: spi@7e630000 {
            compatible = "aspeed,spi-controller";
            reg = <0x7e630000 0xc4>, <0x90000000 0x10000000>;
            reg-names = "ctrl_reg", "spi_mmap";
            clocks = <&sysclk ASPEED_CLK_HCLK>;
            pinctrl-0 = <&pinctrl_spi1_quad>;
            num-cs = <2>;
            ast-platform = <1030>;
            ctrl-type = "hspi";
            label = "SPI1";
            #address-cells = <1>;
            #size-cells = <0>;
            spi-ctrl-caps-mask = <0x000e0c0c>;
            status = "disabled";

            flash@0 {
                compatible = "jedec,spi-nor";
                reg = <0>;
                spi-max-buswidth = <1>;
                spi-max-frequency = <25000000>;
                label = "spi1_cs0";
                status = "disabled";
            };

            flash@1 {
                compatible = "jedec,spi-nor";
                reg = <1>;
                spi-max-buswidth = <1>;
                spi-max-frequency = <25000000>;
                label = "spi1_cs1";
                status = "disabled";
            };
        };

        spi2: spi@7e640000 {
            compatible = "aspeed,spi-controller";
            reg = <0x7e640000 0xc4>, <0xb0000000 0x10000000>;
            reg-names = "ctrl_reg", "spi_mmap";
            clocks = <&sysclk ASPEED_CLK_HCLK>;
            pinctrl-0 = <&pinctrl_spi2_default
                         &pinctrl_spi2_cs1
                         &pinctrl_spi2_quad>;
            num-cs = <2>;
            ast-platform = <1030>;
            ctrl-type = "nspi";
            label = "SPI2";
            #address-cells = <1>;
            #size-cells = <0>;
```

```
            spi-ctrl-caps-mask = <0x000e0c0c>;
            status = "disabled";

            flash@0 {
                compatible = "jedec,spi-nor";
                reg = <0>;
                spi-max-buswidth = <1>;
                spi-max-frequency = <25000000>;
                label = "spi2_cs0";
                status = "disabled";
            };

            flash@1 {
                compatible = "jedec,spi-nor";
                reg = <1>;
                spi-max-buswidth = <1>;
                spi-max-frequency = <25000000>;
                label = "spi2_cs1";
                status = "disabled";
            };
        };
```

```
&fmc {
    status = "okay";
};

&fmc_cs0 {
    status = "okay";
    spi-max-buswidth = <4>;
    spi-max-frequency = <50000000>;
};

&fmc_cs1 {
    status = "okay";
    spi-max-buswidth = <4>;
    spi-max-frequency = <50000000>;
};
```

```
&spi1 {
    status = "okay";
};

&spi1_cs0 {
    status = "okay";
    spi-max-buswidth = <4>;
    spi-max-frequency = <50000000>;
};

&spi1_cs1 {
    status = "okay";
    spi-max-buswidth = <4>;
    spi-max-frequency = <50000000>;
};

&spi2 {
    status = "okay";
};

&spi2_cs0 {
    status = "okay";
    spi-max-buswidth = <4>;
    spi-max-frequency = <50000000>;
};

&spi2_cs1 {
    status = "okay";
    spi-max-buswidth = <4>;
    spi-max-frequency = <50000000>;
};
```

6.17.2.2. DTS property

yaml files	dts/bindings/spi/aspeed,spi-controller.yaml dts/bindings/mtd/jedec,spi-nor.yaml

6.17.3. Usage

6.17.3.1. Shell command

source files	`drivers/flash/flash_shell.c`

Configurations	Descriptions
`CONFIG_FLASH_SHELL`	Enable the SPI NOR flash related shell commands.

- Usage:

```
uart:~$ flash
 - PWM shell commands
Subcommands:
  erase       :[<device>] <page address> [<size>]
  read        :[<device>] <address> [<Dword count>]
  test        :[<device>] <address> <size> <repeat count>
  write       :[<device>] <address> <dword> [<dword>...]
  update_test  :[<device>] <address> [<count>]
uart:~$ flash update_test fmc_cs0 ef100
Writing 4092 bytes to fmc_cs0 (offset: 0x000ef100)...
Update done.
RW test pass
```

6.17.3.2. APIs Introduction

SPI NOR flash updated sample code, cmd_update_test, can be found in drivers/flash/flash_shell.c. The function, static int **do_update**(const struct device flash_device, off_t offset, uint8_t *buf, size_t len), is the core function for flash update purpose.

Also, SPI NOR flash HAL provides the following basic APIs. Developers can utilize them to achieve scenarios for what they want.

/* read flash */

int flash_read(const struct device *dev, off_t offset, void *data, size_t len);

/* write flash */

int flash_write(const struct device *dev, off_t offset, const void *data, size_t len);

/* erase flash */

int flash_erase(const struct device *dev, off_t offset, size_t size);

/* re-probe flash */

int spi_nor_re_init(const struct device *dev);
For some scenarios, BMC cannot access flash through physical path at the early boot stage. Thus, the related SPI driver is failed at initial probe stage which will result in the incorrect result of reading from or writing to flash. This re-probe API allow developer to re-initialize SPI flash driver before using flash read/write APIs.

6.18. I3C driver

Aspeed Zephyr I3C driver implements the JEDEC JESD403-1 initialization procedure and the private transfer protocol.

6.18.1. Driver

6.18.1.1. Source file and configuration

interface	`include/drivers/i3c/i3c.h`

source files	`drivers/i3c/i3c_global_aspeed.c`
	`drivers/i3c/i3c_aspeed.c`
	`drivers/i3c/i3c_common.c`
	`drivers/i3c/slave/i3c_slave_mqueue.c`

Kconfig source files	`drivers/i3c/Kconfig`
	`drivers/i3c/Kconfig.aspeed`
	`drivers/i3c/slave/Kconfig`
	`drivers/i3c/slave/Kconfig.i3c_slave_mqueue`

Configurations	Descriptions

`CONFIG_I3C=y`
`CONFIG_I3C_ASPEED=y`
	Enable Aspeed I3C driver

`CONFIG_I3C_ASPEED_MAX_IBI_PAYLOAD`	The maximum IBI payload size plus the MDB. The minimum value is “2”
`CONFIG_I3C_SLAVE=y`	Enable I3C slave driver
`CONFIG_I3C_SLAVE_INIT_PRIORITY`	the priority of the slave driver initialization. The value shall be greater than “CONFIG_KERNEL_INIT_PRIORITY_DEVICE”
`CONFIG_HEAP_MEM_POOL_SIZE`	Allocate at least 4096 bytes for the I3C driver usage.
`CONFIG_I3C_SLAVE_MQUEUE=y`	Enable I3C slave message queue driver

6.18.1.2. DTS

DT source file	dts/arm/aspeed/ast26xx.dtsi dts/arm/aspeed/ast10x0.dtsi

```
/{
    …
    soc {
        …
        i3c: bus@7e7a0000 {
            compatible = "simple-bus";
            #address-cells = <1>;
            #size-cells = <1>;
            ranges = <0 0x7e7a0000 0x8000>;
        };
        …
    };
};
…
&i3c {
    i3c_gr: i3c-global-regs@0 {
        compatible = "aspeed,i3c-global";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x0 0x100>;
        resets = <&sysrst ASPEED_RESET_I3C>;
        ni3cs = <6>;
        status = "disabled";
    };

    i3c0: i3c0@2000 {
        compatible = "aspeed,i3c";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x2000 0x1000>;
        interrupts = <102 0>;
        resets = <&sysrst ASPEED_RESET_I3C0>;
        clocks = <&sysclk ASPEED_CLK_GATE_I3C0CLK>;
        i2c-scl-hz = <400000>;
        i3c-scl-hz = <12500000>;
        instance-id = <0>;
        pinctrl-0 = <&pinctrl_i3c0_default>;
        status = "disabled";
        label = "I3C_0";
    };

    i3c1: i3c1@3000 {
        compatible = "aspeed,i3c";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x3000 0x1000>;
        interrupts = <103 0>;
        resets = <&sysrst ASPEED_RESET_I3C1>;
        clocks = <&sysclk ASPEED_CLK_GATE_I3C1CLK>;
        i2c-scl-hz = <400000>;
        i3c-scl-hz = <12500000>;
        instance-id = <1>;
        pinctrl-0 = <&pinctrl_i3c1_default>;
        status = "disabled";
        label = "I3C_1";
    };

    i3c2: i3c2@4000 {
        compatible = "aspeed,i3c";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x4000 0x1000>;
        interrupts = <104 0>;
        resets = <&sysrst ASPEED_RESET_I3C2>;
        clocks = <&sysclk ASPEED_CLK_GATE_I3C2CLK>;
        i2c-scl-hz = <400000>;
        i3c-scl-hz = <12500000>;
        instance-id = <2>;
        status = "disabled";
        label = "I3C_2";
    };

    i3c3: i3c3@5000 {
        compatible = "aspeed,i3c";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x5000 0x1000>;
        interrupts = <105 0>;
        resets = <&sysrst ASPEED_RESET_I3C3>;
        clocks = <&sysclk ASPEED_CLK_GATE_I3C3CLK>;
        i2c-scl-hz = <400000>;
        i3c-scl-hz = <12500000>;
        instance-id = <3>;
        status = "disabled";
        label = "I3C_3";
    };

    i3c4: i3c4@6000 {
        compatible = "aspeed,i3c";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x6000 0x1000>;
        interrupts = <106 0>;
        resets = <&sysrst ASPEED_RESET_I3C4>;
        clocks = <&sysclk ASPEED_CLK_GATE_I3C4CLK>;
        i2c-scl-hz = <400000>;
        i3c-scl-hz = <12500000>;
        instance-id = <4>;
        pinctrl-0 = <&pinctrl_hvi3c4_default>;
        status = "disabled";
        label = "I3C_4";
    };

    i3c5: i3c5@7000 {
        compatible = "aspeed,i3c";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x7000 0x1000>;
        interrupts = <107 0>;
        resets = <&sysrst ASPEED_RESET_I3C5>;
        clocks = <&sysclk ASPEED_CLK_GATE_I3C5CLK>;
        i2c-scl-hz = <400000>;
        i3c-scl-hz = <12500000>;
        instance-id = <5>;
        pinctrl-0 = <&pinctrl_hvi3c5_default>;
        status = "disabled";
        label = "I3C_5";
    };
};
```

6.18.1.3. DT bindings

yaml files	`dts/bindings/i3c/i3c-controller.yaml`
	`dts/bindings/i3c/aspeed,i3c.yaml`
	`dts/bindings/i3c/aspeed,i3c-global.yaml`
	`dts/bindings/i3c/aspeed,i3c-slave-mqueue.yaml`

6.18.1.4. API list

Function	i3c_master_attach_device
Brief	Attach the slave device to a bus
Parameters	dev – the i3c master device of the I3C bus
	slave – the slave device descriptor to be attached
Return	0 if success

Function	i3c_master_detach_device
Brief	Detach the slave device from the bus
Parameters	dev – the i3c master device of the I3C bus
	slave – the slave device descriptor to be detached
Return	0 if success

Function	i3c_master_send_ccc
Brief	master sends CCC to the bus
Parameters	dev – the i3c master device of the I3C bus
	ccc – the CCC structure

Return	0 if success

Function	i3c_master_priv_xfer
Brief	master sends private transfer to the specified slave device
Parameters	i3cdev – the slave device descriptor
	xfers – the pointer to the private transfers
	nxfers – number of the private transfers
Return	0 if success

Function	i3c_master_request_ibi
Brief	Requests IBI from the specified slave device. The IBI is registered but not enabled yet.
Parameters	i3cdev – the slave device descriptor
	cb – the callback function and arguments when an IBI occured
Return	0 if success

Function	i3c_master_enable_ibi
Brief	enable IBI for the specified slave device
Parameters	i3cdev – the slave device descriptor
Return	0 if success

Function	i3c_i2c_read
Brief	read data from the I2C slave device
Parameters	slave – the slave device descriptor
	addr – address to be read
	buf – buffer of the read data
	length – length of the read data
Return	0 if success

Function	i3c_i2c_write
Brief	write data to the I2C slave device
Parameters	slave – the slave device descriptor
	addr – address to be write
	buf – buffer of the write data
	length – length of the write data
Return	0 if success

Function	i3c_jesd403_read
Brief	read data from the JESD403 compliant slave device
Parameters	slave – the slave device descriptor
	addr – pointer to the address data
	buf – buffer of the read data
	length – length of the read data
Return	0 if success

Function	i3c_jesd403_write
Brief	write data to the JESD403 compliant slave device
Parameters	slave – the slave device descriptor
	addr – pointer to address data
	buf – buffer of the write data
	length – length of the write data
Return	0 if success

The funtions below are used when the I3C controller is in the slave mode.

Function	i3c_slave_register
Brief	register the I3C controller in the slave mode
Parameters	dev – the I3C controller device
	slave_data – the slave data, including the pointer to the device and callback function pointers.
Return	0 if success

Function	i3c_slave_send_sir
Brief	Send slave-interrupt-request
Parameters	dev – the I3C controller device
	mdb – mandatory data byte
	data – pointer to the data to be sent
	nbytes – number of data bytes to be sent
Return	0 if success

Function	i3c_slave_prep_read_data
Brief	Prepare the data to be read from the bus master
Parameters	dev – the I3C controller device
	data – pointer to the data to be read
	nbytes – number of data bytes to be read

	wait – 1 = wait for data read done. 0 = no wait
Return	0 if success

Function	i3c_aspeed_slave_wait_data_consume
Brief	Wait for the read data consume (read by the master). Call the function when called i3c_slave_prep_read_data with wait flag = 0
Parameters	dev – the I3C controller device
Return	0 if success

6.18.2. Samples

6.18.2.1. I3C0 and I3C1 loopback

source files	`samples/drivers/i3c/`

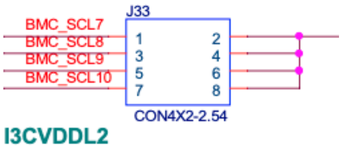
Kconfig source files	`samples/drivers/i3c/prj.conf`
	`samples/drivers/i3c/Kconfig`
	`samples/drivers/i3c/boards/ast1030_evb.conf`
	`samples/drivers/i3c/boards/ast2600_evb.conf`
	`samples/drivers/i3c/overlay-loopback.conf`

DTS overlay	`samples/drivers/i3c/loopback.overlay`
	`samples/drivers/i3c/boards/ast1030_evb.overlay`
	`samples/drivers/i3c/boards/ast2600_evb.overlay`

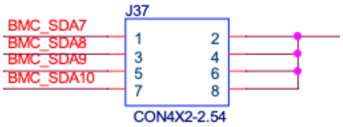
build command	`west build -p auto -b ast2600_evb \`
	`samples/drivers/i3c \`
	`-DDTC_OVERLAY_FILE="boards/ast2600_evb.overlay;\`
	`loopback.overlay" \`
	`-DOVERLAY_CONFIG="overlay-loopback.conf"`
	
	`west build -p auto -b ast1030_evb \`
	`samples/drivers/i3c \`
	`-DDTC_OVERLAY_FILE="boards/ast1030_evb.overlay;\`
	`loopback.overlay" \`
	`-DOVERLAY_CONFIG="overlay-loopback.conf"`

- AST1030 EVB setup

Connect BMC_SCL7 to BMC_SCL8



Connect BMC_SDA7 to BMC_SDA8



- Expected result

```
*** Booting Zephyr OS build v00.01.00-208-g8ac09a4e6cd6  ***
slave pid = 7ec80001000
bus init done

bf 5c
loopback test pass
```

6.18.2.2. AST2600 I3C master to AST1030 I3C slave

source files	drivers/i3c/i3c_shell.c
Kconfig source files	drivers/i3c/Kconfig
DTS overlay	samples/boards/ast1030_evb/demo/boards/ ast1030_evb.overlay
build command	west build -p auto -b ast1030_evb samples/boards/ast1030_evb/demo

- AST2600 Linux configuration

Linux tag version v00.04.07 or later, modify the DT source file as:

```
/* High-voltage I3C 3.3/1.8V */
&i3c2 {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hvi3c3_default>;
        ast1030: bic@0,7ec80001000 {
                compatible = "i3c-ibi-mqueue";
                reg = <0x0 0x7ec 0x80001000>;
        };
}
```

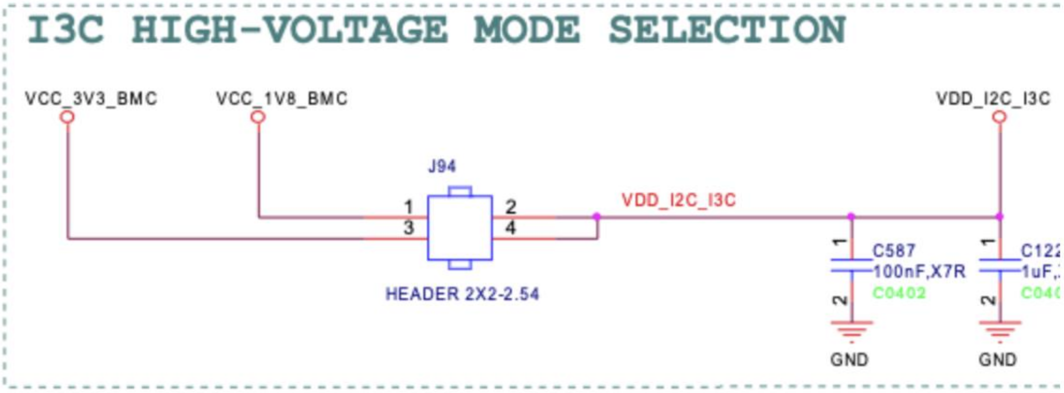
- Board connection

Connect the SCL and SDA pins on AST2600 EVB with the pins on AST1030 EVB. Ground pins also need to be connected.

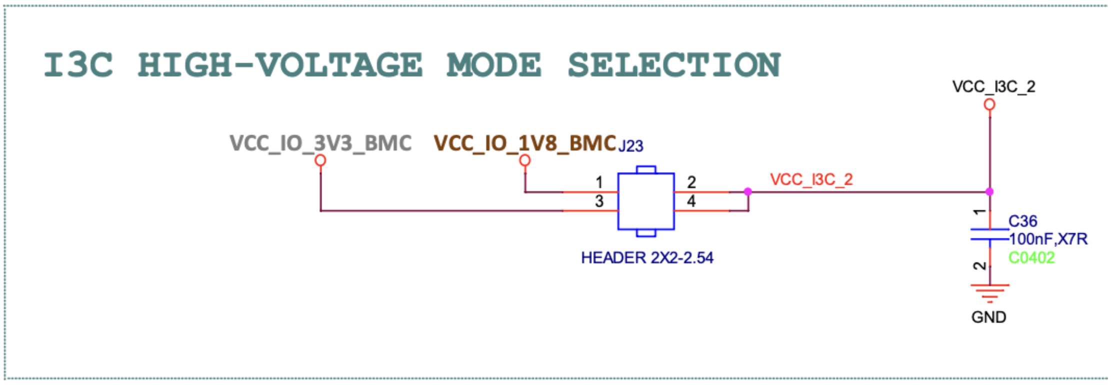
	AST2600 EVB	AST1030 EVB
SCL pin	SCL1_I3C1CLK (HVI3C3SCL)	BMC_SCL8 (HVI3C1SCL)
SDA pin	SCL1_I3C1DATA (HVI3C3SDA)	BMC_SDA8 (HVI3C1SDA)

Select 3.3V for I3C high-voltage mode selection

AST2600 – connect J94 pin3 and pin4



AST1030 – connect J23 pin3 and pin4



- Expected result

AST2600 I3C master sends data to AST1030 slave through the private read command

```
[AST /]# i3ctransfer -d /dev/i3c-2-7ec80011000 -w 1,2,3,4,5
Success on message 0
```

```
uart:~$ i3c smq I3C_1_SMQ -r 5
```

```
00000000: 01 02 03 04 05                                  |.....           |
```

AST1030 I3C slave sends data to AST2600 master through the IBI

```
uart:~$ i3c smq I3C_1_SMQ -w 0xa,0xb,0xc
```

```
[AST /]# hexdump -C /sys/bus/i3c/devices/2-7ec80011000/ibi-mqueue
00000000  0a 0b 0c                                         |...|
00000003
```

6.18.2.3. IMX3102 access (AST2600 only)

source files	samples/drivers/i3c/
Kconfig source files	samples/drivers/i3c/prj.conf samples/drivers/i3c/Kconfig samples/drivers/i3c/boards/ast2600_evb.conf samples/drivers/i3c/overlay-imx3102.conf
DTS overlay	samples/drivers/i3c/boards/ast2600_evb.overlay
build command	west build -p auto -b ast2600_evb samples/drivers/i3c -DOVERLAY_CONFIG="overlay-imx3102.conf"

- Expected result

```
*** Booting Zephyr OS build v00.01.00-159-g5fd940424df4  ***
device ID in I2C mode 31 02
device ID in I3C mode 31 02
```

6.19. Crypto driver

Aspeed Zephyr crypto driver implements the hash, ecdsa and rsa crypto algorithm.

6.19.1. Driver

6.19.1.1. Source file and configuration

interface	`include/crypto/hash.h`
	`include/crypto/rsa.h`
	`include/crypto/ecdsa.h`

source files	`drivers/crypto/hace_aspeed.c`
	`drivers/crypto/hash_aspeed.c`
	`drivers/crypto/rsa_aspeed.c`
	`drivers/crypto/ecdsa_aspeed.c`

Kconfig source files	`drivers/crypto/Kconfig` `drivers/crypto/Kconfig.aspeed`

Configurations	Descriptions

`CONFIG_CRYPTO=y`
`CONFIG_CRYPTO_ASPEED=y`
	Enable Aspeed crypto driver

`CONFIG_CRYPTO=y`
`CONFIG_RSA_ASPEED=y`
	Enable Aspeed rsa driver

`CONFIG_CRYPTO=y`
`CONFIG_ECDSA_ASPEED=y`
	Enable Aspeed ecdsa driver

6.19.1.2. API list

Function	hash_begin_session
Brief	Setup a hash session
Parameters	dev - Pointer to the device structure for the driver instance.
	ctx - Pointer to the context structure.
	algo - The hash algorithm to be used in this session. e.g SHA512
Return	0 if success

Function	hash_free_session
Brief	Cleanup a hash session
Parameters	dev - Pointer to the device structure for the driver instance.

	ctx - Pointer to the context structure.
Return	0 if success

Function	hash_update
Brief	Perform Hash update.
Parameters	ctx - Pointer to the hash context of this op.
	pkt - Structure holding the input/output buffer pointers.
Return	0 if success

Function	hash_final
Brief	Perform Hash final.
Parameters	ctx - Pointer to the hash context of this op.
	pkt - Structure holding the input/output buffer pointers.
Return	0 if success

Function	rsa_begin_session
Brief	Setup a rsa session
Parameters	dev - Pointer to the device structure for the driver instance.
	ctx - Pointer to the context structure.
	key - Pointer to the rsa key structure.
Return	0 if success

Function	rsa_free_session
Brief	Cleanup a rsa session
Parameters	dev - Pointer to the device structure for the driver instance.
	ctx - Pointer to the context structure.
Return	0 if success

Function	rsa_encrypt
Brief	Perform RSA encrypt.
Parameters	ctx - Pointer to the hash context of this op.
	pkt - Structure holding the input/output buffer pointers.
Return	0 if success

Function	rsa_decrypt
Brief	Perform RSA decrypt.

Parameters	ctx - Pointer to the hash context of this op.
	pkt - Structure holding the input/output buffer pointers.
Return	0 if success

Function	rsa_sign
Brief	Perform RSA sign.
Parameters	ctx - Pointer to the hash context of this op.
	pkt - Structure holding the input/output buffer pointers.
Return	0 if success

Function	rsa_verify
Brief	Perform RSA verify.
Parameters	ctx - Pointer to the hash context of this op.
	pkt - Structure holding the input/output buffer pointers.
Return	0 if success

Function	ecdsa_begin_session
Brief	Setup a ecdsa session
Parameters	dev - Pointer to the device structure for the driver instance.
	ctx - Pointer to the context structure.
	key - Pointer to the rsa key structure.
Return	0 if success

Function	ecdsa_free_session
Brief	Cleanup a ecdsa session
Parameters	dev - Pointer to the device structure for the driver instance.
	ctx - Pointer to the context structure.
Return	0 if success

Function	ecdsa_verify
Brief	Perform ECDSA verify.
Parameters	ctx - Pointer to the hash context of this op.
	pkt - Structure holding the input/output buffer pointers.
Return	0 if success

6.19.2. Device Tree Source

6.19.2.1. Enabling the hace and rsa device

DT source file	dts/arm/aspeed/ast10x0.dtsi

```
hace: hace@7e6d0000 {
    compatible = "aspeed,hace";
    #address-cells = <1>;
    #size-cells = <1>;
    reg = <0x7e6d0000 0x200>;
    clocks = <&sysclk ASPEED_CLK_GATE_YCLK>;
    resets = <&sysrst ASPEED_RESET_HACE>;
};

rsa: rsa@7e6f2000 {
    compatible = "aspeed,rsa";
    #address-cells = <1>;
    #size-cells = <1>;
    reg = <0x7e6f2000 0x100
             0x79000000 0x1800>;
    clocks = <&sysclk ASPEED_CLK_GATE_RSACLK>;
    label = "RSA";
};
```

6.19.2.2. DT bindings

yaml files	dts/bindings/crypto/aspeed,hace.yaml dts/bindings/crypto/aspeed,rsa.yaml

6.19.3. Usage

6.19.3.1. Shell command

source files	drivers/crypto/hash_shell.c drivers/crypto/rsa_shell.c

Configurations	Descriptions
CONFIG_HASH_SHELL	Enable the HASH test shell command.
CONFIG_RSA_SHELL	Enable the RSA test shell command.

-Usage

```
uart:~$ hash test
sha256_test
tv[0]:PASS
tv[1]:PASS
tv[2]:PASS
tv[3]:PASS
tv[4]:PASS
sha384_test
tv[0]:PASS
tv[1]:PASS
tv[2]:PASS
tv[3]:PASS
tv[4]:PASS
tv[5]:PASS
sha512_test
tv[0]:PASS
tv[1]:PASS
tv[2]:PASS
tv[3]:PASS
tv[4]:PASS
tv[5]:PASS

uart:~$ rsa test
rsa test vector[0]:
enc pass
dec pass
rsa test vector[1]:
enc pass
dec pass
rsa test vector[2]:
enc pass
dec pass
```

6.20. Watchdog Timer (WDT)

6.20.1. Source files and configurations

interface	include/drivers/watchdog/watchdog.h
source files	drivers/watchdog/wdt_aspeed.c
Kconfig source files	drivers/watchdog/Kconfig drivers/watchdog/Kconfig.aspeed

Configurations	Descriptions
CONFIG_WATCHDOG=y	Watchdog support
CONFIG_WDT_ASPEED=y	Enable ASPEED WDT controller driver.

6.20.2. DTS

6.20.2.0. Enable SPI flash device.

.dtsi:

```
        wdt0: wdt_common@7e785000 {
            compatible = "aspeed,ast-watchdog";
            interrupts = <INTR_WDT AST10X0_IRQ_DEFAULT_PRIORITY>;
            reg = <0x7e785000 0x200>;
            #address-cells = <1>;
            #size-cells = <0>;
            label = "wdt0";
            status = "disabled";

            wdt1: wdt@1 {
                reg = <1>;
                aspeed,scu = <&syscon>;
                reset-mask= <0x30f1ff1 0x03fffff1>;
                label = "wdt1";
                status = "disabled";
            };

            wdt2: wdt@2 {
                reg = <2>;
                aspeed,scu = <&syscon>;
                reset-mask= <0x30f1ff1 0x03fffff1>;
                label = "wdt2";
                status = "disabled";
            };

            wdt3: wdt@3 {
                reg = <3>;
                aspeed,scu = <&syscon>;
                reset-mask= <0x30f1ff1 0x03fffff1>;
                label = "wdt3";
                status = "disabled";
            };

            wdt4: wdt@4 {
                reg = <4>;
                aspeed,scu = <&syscon>;
                reset-mask= <0x30f1ff1 0x03fffff1>;
                label = "wdt4";
                status = "disabled";
            };
        };
```

.dts:

```
&wdt0 {
    status = "okay";
};

&wdt1 {
    status = "okay";
};

&wdt2 {
    status = "okay";
};

&wdt3 {
    status = "okay";
};

&wdt4 {
    status = "okay";
};
```

6.20.2.1. DTS property

yaml files	dts/bindings/watchdog/aspeed,ast-watchdog.yaml

6.20.3. Usage

6.20.3.0. API Definitions

Name	**wdt_install_timeout**
Description	Configure timeout period and callback function.
Arguments	
dev	Watchdog device node.
cfg	**window.min:** Should be 0 for AST1030. **window.max:** Timeout period. **window.callback:** Callback function executed in ISR after timeout occurs. If NULL, it will be ignored in ISR.
Return value	**0:** Successful, without error. **Non-zero:** Error occurs.

Name	**wdt_setup**
Description	Enable watchdog timer
Arguments	
dev	Watchdog device node.
options	**WDT_FLAG_RESET_NONE:** Don’t reset the device, only ISR is executed. **WDT_FLAG_RESET_CPU_CORE:** Trigger SoC reset after timeout.

	WDT_FLAG_RESET_SOC: Trigger full chip reset after timeout.
Return value	**0:** Successful, without error. **Non-zero:** Error occurs.

Name	**wdt_feed**
Description	Feed/restart a specified watchdog timer.
Arguments	
dev	Watchdog device node.
channel_id	Unused currently in ASPEED WDT driver, wdt_aspeed.c.
Return value	**0:** Successful, without error. **Non-zero:** Error occurs.

Name	**wdt_disable**
Description	Disable watchdog instance.
Arguments	
dev	Watchdog device node.
Return value	**0:** Successful, without error. **Non-zero:** Error occurs.

6.20.3.1. Shell Commands

source files	soc/arm/aspeed/ast10x0/soc.c

Command	**kernel reboot warm**
Description	Trigger SoC reset

Command	**kernel reboot cold**
Description	Trigger full chip reset

6.20.4. Demo

6.20.4.0. Sample Code

There exists two sample code for AST1030 WDT.

samples/boards/ast1030_evb/demo/src/demo_wdt.c

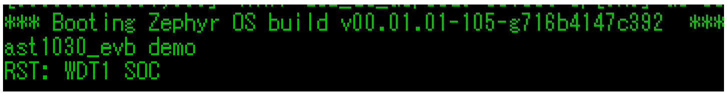
6.20.4.1. Demo Scenarios

The sample code mainly demonstrates how to use WDT APIs. In AST030_evb demo scenario, WDT2 is the background WDT. It is fed/restarted periodically.

If the number of HW WDT, four, is insufficient, user or developer can utilize single HW WDT to monitor multiple tasks. Please refer to subsys/task_wdt/task_wdt.c. The shortest timeout period of overall tasks is set to HW WDT when wdt_feed function is executed. Also, the callback function registered to each task can be executed after timeout occurs. Notice, this is a sample code, developer needs to study and modify it in order to satisfy the real scenario.

6.20.4.2. Boot reason

After power on or reboot, boot reason will be shown.



```
*** Booting Zephyr OS build v00.01.01-105-g716b4147c392  ***
ast1030_evb demo
RST: WDT1 SOC
```

6.20.4.3. Source file and configuration

interface	`soc/arm/aspeed/ast10x0/soc.h`
source files	`soc/arm/aspeed/ast10x0/soc.c`

Function: aspeed_print_sysrst_info()

6.21. Timer driver

Aspeed timer driver provides the interface to access the Aspeed timers on the APB bus. The hardware supports the clock source be APB clock 50MHz or 1MHz, but the for the moment only 50MHz is implemented.   Besides, the driver only supports for AST10x0 series SOC for now.

6.21.1. Driver

6.21.1.1. Source file and configuration

interface	include/drivers/timer/aspeed_timer.h
source files	drivers/timer/aspeed_timer.c drivers/timer/aspeed_timer_shell.c
Kconfig source files	drivers/timer/Kconfig

Configurations	Descriptions
CONFIG_TIMER_ASPEED	Enable Aspeed timer driver
CONFIG_TIMER_ASPEED_SHELL	Enable the shell commands of the Aspeed timer.

6.21.1.2. Enabling the timer devices

DT source file	include/drivers/timer/aspeed_timer.h

```
/{
soc {
  …
  timers: timers@7e782000 {
    compatible = "aspeed,timers";
    reg = <0x7e782000 0x100>;
    timer0: timer0 {
      compatible = "aspeed,timer";
      interrupts = <16 AST10X0_IRQ_DEFAULT_PRIORITY>;
      clocks = <&sysclk ASPEED_CLK_PCLK>;
      label = "TIMER0";
      index = <0>;
      status = "disabled";
    };
    …
    timer7: timer7 {
      compatible = "aspeed,timer";
      interrupts = <23 AST10X0_IRQ_DEFAULT_PRIORITY>;
      clocks = <&sysclk ASPEED_CLK_PCLK>;
      label = "TIMER7";
      index = <7>;
      status = "disabled";
    };
  };
};
```

6.21.1.3. DT bindings

yaml files	`dts/bindings/timer/aspeed,timer.yaml`

6.21.1.4. API list

Struct	aspeed_timer_user_config
Brief	structure for timer configuration
members	millisec millisecond to be waiting for
	callback callback function when timer expired
	user_data context of the callback function
	timer_type: ASPEED_TIMER_TYPE_ONE_SHOT – one-shot timer ASPEED_TIMER_TYPE_PERIODIC – periodic timer

Function	timer_aspeed_start
Brief	Start the timer
Parameters	dev – specify the timer device
	user_config – see aspeed_timer_user_config
Return	0 if success

Function	timer_aspeed_stop
Brief	Stop the timer
Parameters	dev – specify the timer device
Return	0 if success

Function	timer_aspeed_query
Brief	query the timer counter
Parameters	dev – specify the timer device
Return	current timer counter, 0 indicates the timer is expired

6.21.2. Samples

source files	drivers/timer/aspeed_timer_shell.c
Kconfig source files	drivers/timer/Kconfig
configurations	CONFIG_TIMER_ASPEED_SHELL

command	# start a timer timer start <dev> -p <period> -t <type>
arguments	dev – TIMER0, TIMER1, …, TIMER7
	period – milliseconds, in decimal format
	type – 0=one-shot, 1=periodic
example	# start a one-shot timer on timer0 with period 1000ms timer start TIMER0 -p 1000 -t 0 # start a periodic timer on timer 1 with period 3000ms timer start TIMER1 -p 3000 -t 1

command	# stop a timer timer stop <dev>
arguments	dev – TIMER0, TIMER1, …, TIMER7
example	# stop timer0 timer stop TIMER0 # stop timer1 timer stop TIMER1

command	# query the current counter of a timer timer query <dev>
arguments	dev – TIMER0, TIMER1, …, TIMER7
example	# query the current counter of timer0 timer query TIMER0

	# query the current counter of timer1 timer query TIMER1

7. Debug

7.1. UART console

The default UART indices used for different EVBs are listed below.

Board	default UART (HW index)	UART pins
AST2600 EVB	UART 11	TXD11: P23 RXD11: T24
AST1030 EVB	UART 5	TXD5: D13 RXD5: A14

To change the default UART, modify the board dts file, for example, `boards/arm/ast1030_evb/ast1030_evb.dts`. Or overwrite the variables: `zephy,console` and `zephyr,shell-uart` in your project overlay file.

```
chosen {
         zephyr,console = &uart5;
         zephyr,shell-uart = &uart5;
         zephyr,sram = &sram0;
     };
```

7.1.1. Memory commands

- md: dump N * 32-bit double words from the specified memory address

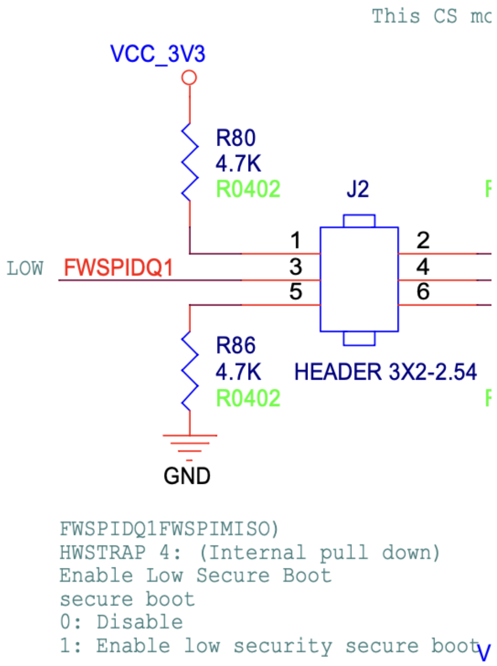
```
md <address> <number of 32-bit double word>
```

- mw: write a single 32-bit double word to the specified memory address

```
mw <address> <value>
```

7.2. Hardware debug UART

To validate the HW debug UART on AST1030 EVB, please enable the low security boot strap (HWSTRAP 4)



7.3. JTAG ICE

To use JTAG ICE debug for ASPEED EVBs (ASPEED SOCs play the role as the JTAG slaves), please enable the configuration `CONFIG_ARM_ICE`.

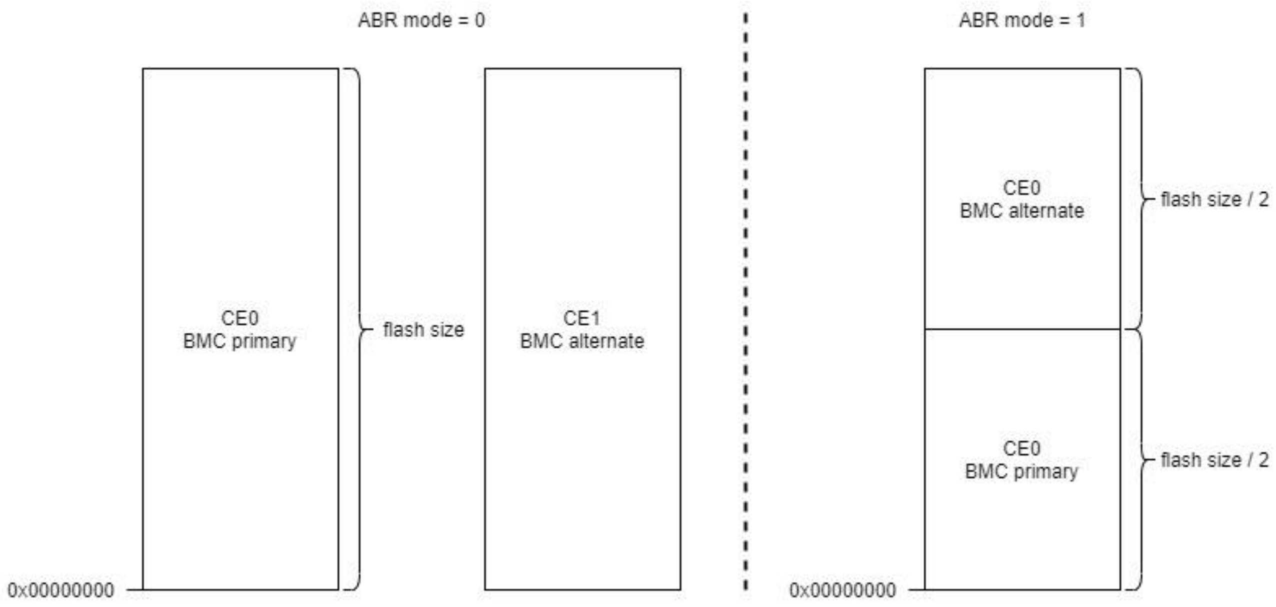
Board	AST1030-EVB
source files	soc/arm/aspeed/ast10x0/sig_def_list.h boards/arm/ast1030_evb/fun_def_list.h
Kconfig source files	drivers/pinmux/Kconfig.aspeed

Board	AST2600-EVB
source files	soc/arm/aspeed/ast26xx/sig_def_list.h boards/arm/ast2600_evb/fun_def_list.h
Kconfig source files	drivers/pinmux/Kconfig.aspeed

Configurations	Descriptions
CONFIG_ARM_ICE	ARM ICE debug enabling

8. Alternate Bootblock Recovery (ABR)

For a stable system, the mechanism for boot image backup is necessary. ABR feature is introduced to avoid system being bricked. Once the primary boot image is tampered by malware or because of component ageing, the backup (alternate) boot image can be used to boot up system and potentially, recovery the primary image. For AST1030, there are two modes for FMC/SPI1 ABR, two flashes (ABR mode = 0) and single flash (ABR mode = 1), as fig. abr-f-1.



▲Figure abr-f-1

The start address of CE0 flash in fig. abr-f-1 is 0x00000000 which is the address view from a physical flash. For accessing flashes through FMC or SPI controller, please use the address space from BMC view. The valid flash decoding address is as below, and the total decode space for each controller is 256MB.

FMC: 0x80000000 – 0x8FFFFFFF
SPI1: 0x90000000 – 0x9FFFFFFF
SPI2: 0xB0000000 – 0xBFFFFFFF

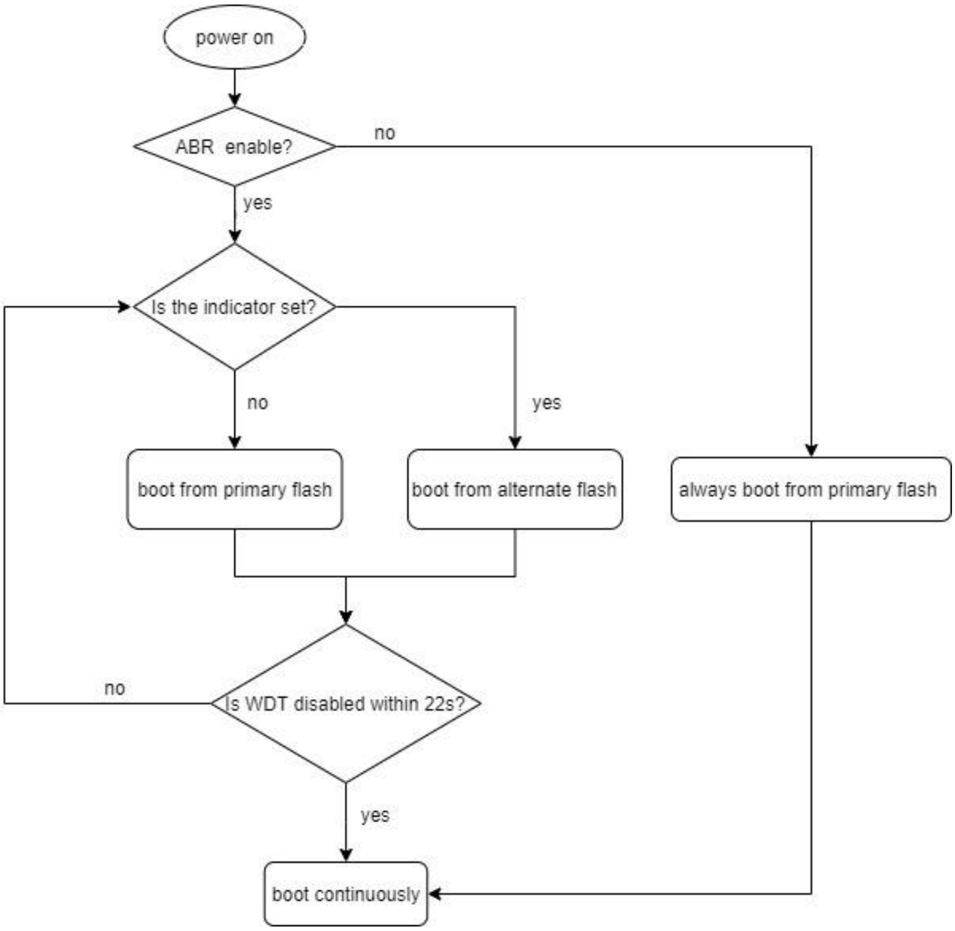
In the following sections, FMC ABR mode with two flashes scenario will be described in detail first, then, single flash case. After that, FW update or primary boot image recovery process will be introduced. Finally, host SPI1 ABR feature is described.
For the naming rule, “FMC#num” or “SPIR#num” is the register name, please refer to FMC and SPI controller sections of AST1030 datasheet.

8.1. FMC ABR with Two Flashes

For two flashes scenario, WDT is triggered automatically after power on or reset, and the primary flash (CE0) is adopted at the first boot time. If WDT is not stopped by FW within specific period (**22s** by default), BMC will be rested and the alternate flash (CE1) will be used to boot up system. Table abr-t-1 shows how to enable FMC two flashes ABR mode and the boot flow diagram is illustrated at fig. abr-f-2.

Platform	Conditions
AST1030	OTPSTRAP[43] = 1 (trap_en_bspiabr) OTPSTRAP[44] = 0 (trap_bspi_abrmode) OTPSTRAP[47:45] (trap_bspi_size) (optional) - 3b’000: flash size is configured by FW (FMC30/34/38) - 3b’001: 1MB - 3b’010: 2MB - 3b’011: 4MB - 3b’100: 8MB - 3b’101: 16MB - 3b’110: 32MB - 3b’111: 64MB

▲Table abr-t-1



▲Figure abr-f-2

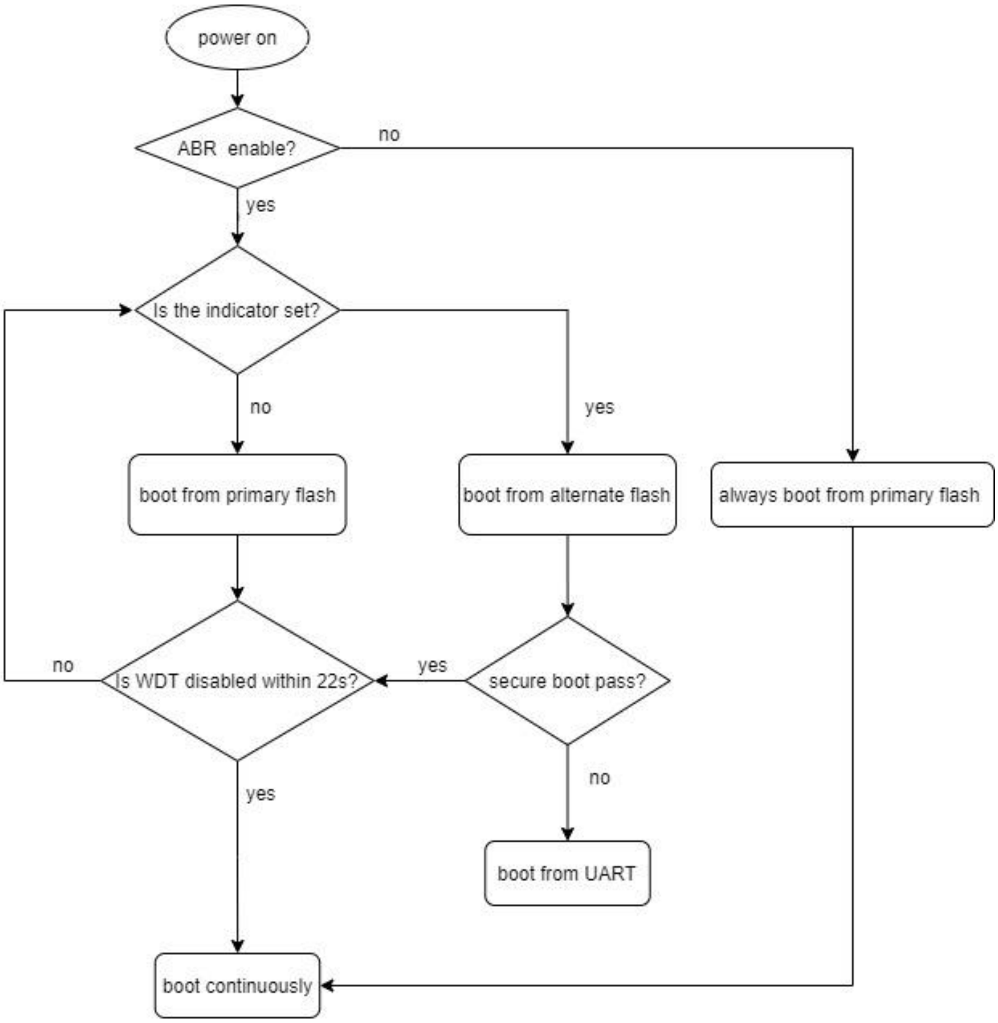
In figure abr-f-2, WDT is triggered automatically after power-on when FMC ABR feature is enabled. **AST1030 FMC_WDT2 to implement FMC ABR**. Indicator is set or cleared whenever WDT timeout. The initial value of indicator is zero and is set after first WDT timeout. Then, BMC uses the alternate flash to boot up system. If WDT is still not disabled, indicator will be cleared after second WDT timeout and boot flash will be switched from the alternate one to the primary one. Therefore, if the BMC system boots up successfully, **remember to disable WDT by FW within 22s**. Table abr-t-2 summarizes the characteristics of WDT and indicator for AST1030.

	AST1030
WDT name / control register	FMC_WDT2 / 0x7e620064
Time out period after reset	22 second (default)
How to disable WDT?	clear 0x7e620064[0]
FMC ABR indicator location	0x7e620064[4]
How to clear indicator?	write 0xea to 0x7e620064[23:16]

▲Table abr-t-2

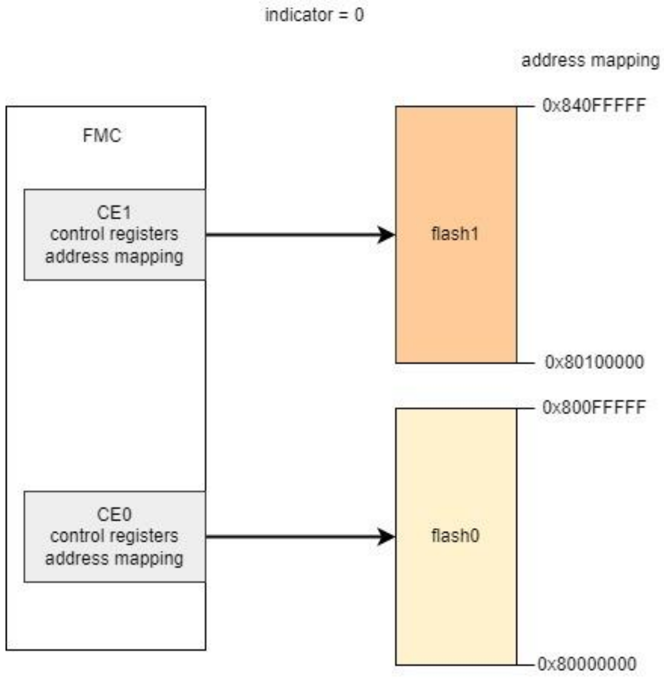
When **secure boot feature is enabled** on AST1030-A1 (or after) and the following two conditions are matched, device will trigger boot from UART mechanism, as figure abr-f-3. Be notice, boot from UART external strap is not needed for this scenario.

- MCU ROM fails to verify image on SRAM.
- FMC ABR indicator is set.



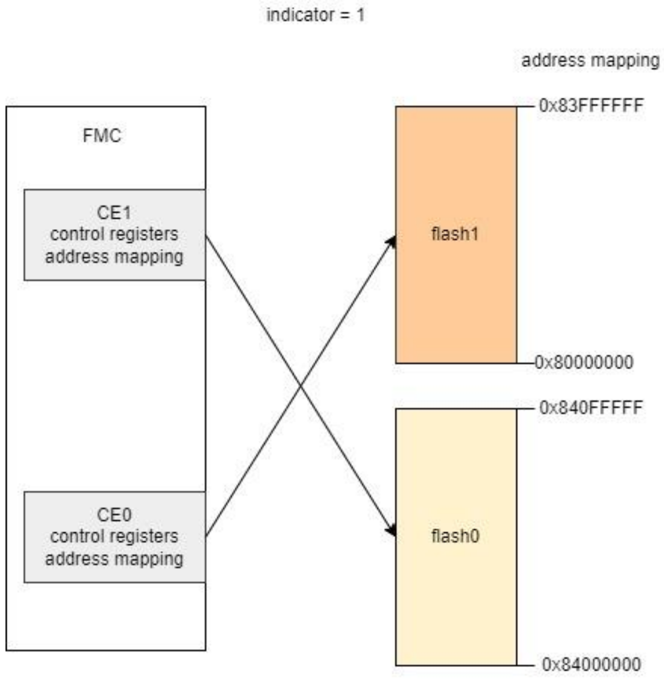
▲Figure abr-f-3

As we known, normally, the address mapping for CE0 and CE1 is controlled by FMC30 and FMC34. For AST1030, CE0 flash (flash0) size is 1MB and assume CE1 flash (flash1) size is 64MB, the initial address mapping of both flashes is shown as fig. abr-f-4.



▲Figure abr-f-4

When power-on, the indicator is zero and the address mapping of flash0 starts from beginning of FMC address mapping, namely, 0x80000000. The address mapping of flash1, which can be configured by FMC34, is concatenated after flash0 in this example. However, when FMC ABR is enabled and BMC is reset by WDT of ABR, the indicator is set and the situation is changed, as fig. abr-f-5.



▲Figure abr-f-5

Obviously, from the FMC controller's view, the address mapping for flash0 and flash1 is swapped. The image of flash1 is used to boot up BMC system now. As mentioned previously, if WDT is still not stopped, the situation will be changed from fig. abr-f-5 to fig. abr-f-4 after next WDT reset. FW developers or device maintainers should keep this mechanism in mind during executing FW update process. The detail about FW update with FMC ABR enabled will be discussed later.

Notice: For AST1030-A0/A1 with dual flashes ABR case, do NOT let the alternate flash (flash0) enter 4-byte address mode during system boots from the primary flash (flash0). Otherwise, system cannot boot from the alternate one if FMC_WDT2 is not stopped within 22 seconds due to wrong address mode between BMC and flash.

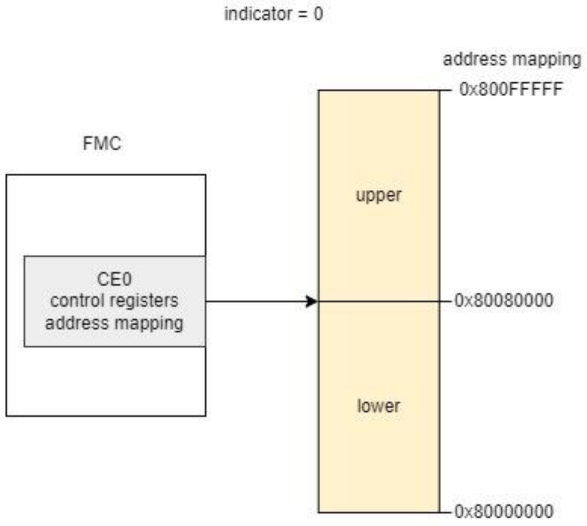
8.2. FMC ABR with Single Flash

Due to some reasons, FMC ABR with single flash mode is usually adopted which concept is like FMC two flashes ABR mode. Table abr-t-3 lists the HW settings in order to enable FMC single flash ABR mode (ABR mode = 1).

Platform	Conditions
AST1030	OTPSTRAP[43] = 1 (trap_en_bspiabr) OTPSTRAP[44] = 1 (trap_bspi_abrmode) OTPSTRAP[47:45] (trap_bspi_size) = 3b'001 (**mandatory**) - 3b'000: flash size is configured by FW (FMC30/34/38) - 3b'001: 1MB - 3b'010: 2MB - 3b'011: 4MB - 3b'100: 8MB - 3b'101: 16MB - 3b'110: 32MB - 3b'111: 64MB

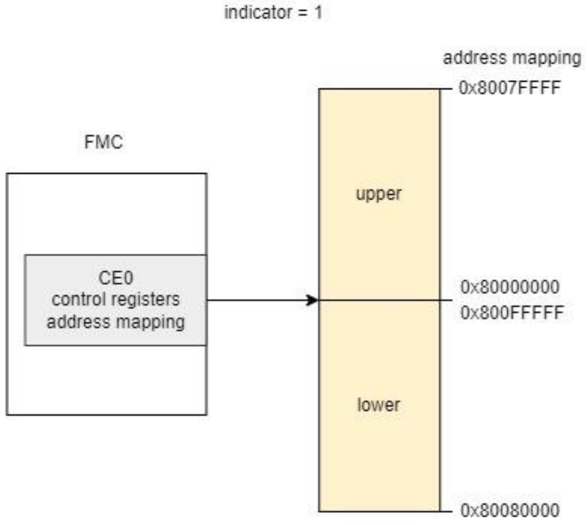
▲Table abr-t-3

The major difference between single flash ABR mode and two flash ABR mode for FMC is that in single flash case, FMC ABR mechanism is implemented independently by CE0 only. Under this situation, a flash is divided into two equal parts. For convenience, we label "upper" and "lower" for these two parts, refer to fig. abr-f-6 (flash size is 1MB). The size of each part is the half of total flash size. Accordingly, total flash size must be determined by related OTP bits in advance.



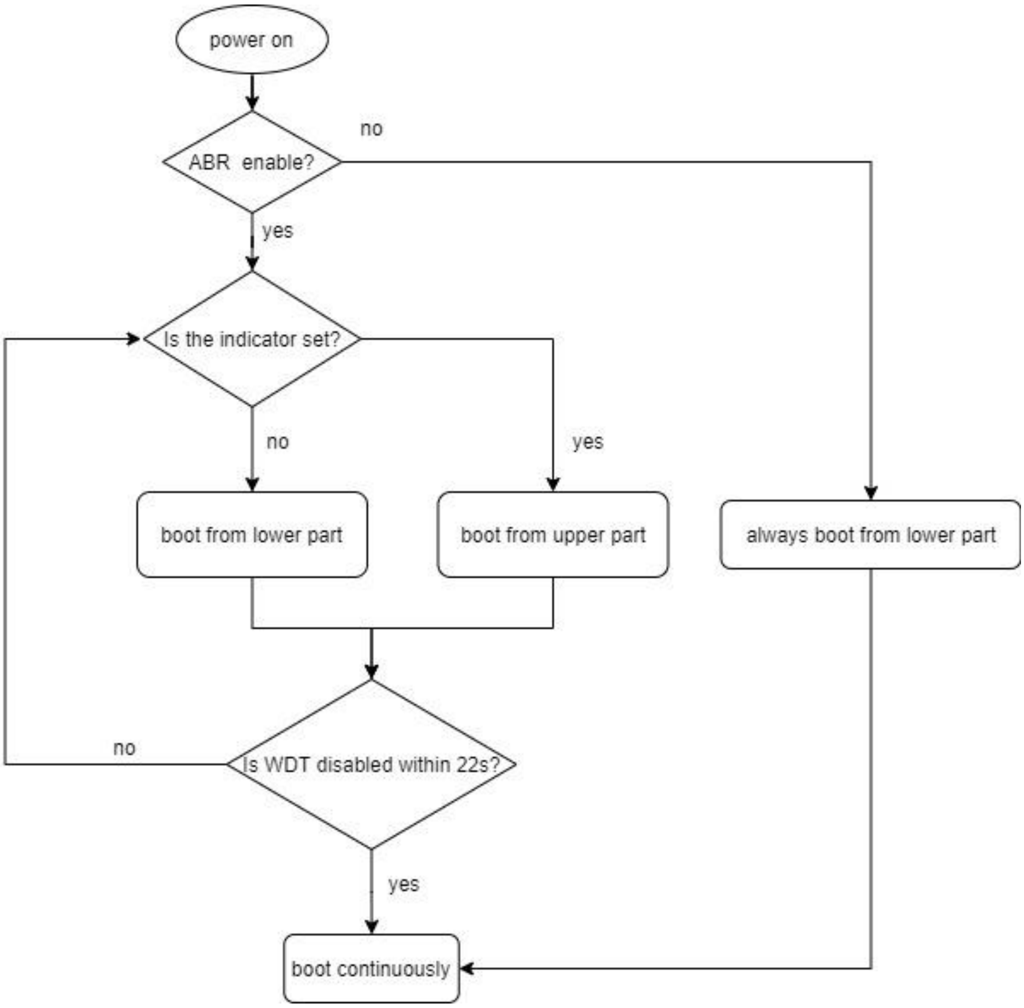
▲Figure abr-f-6

When WDT for ABR timeout occurs, the address mapping for physical flash upper part and lower part is swapped, shown in fig. abr-f-7.



▲Figure abr-f-7

FMC single flash ABR mode’s behavior can be summarized in fig. abr-f-8 by modifying fig. abr-f-2 slightly.



▲Figure abr-f-9

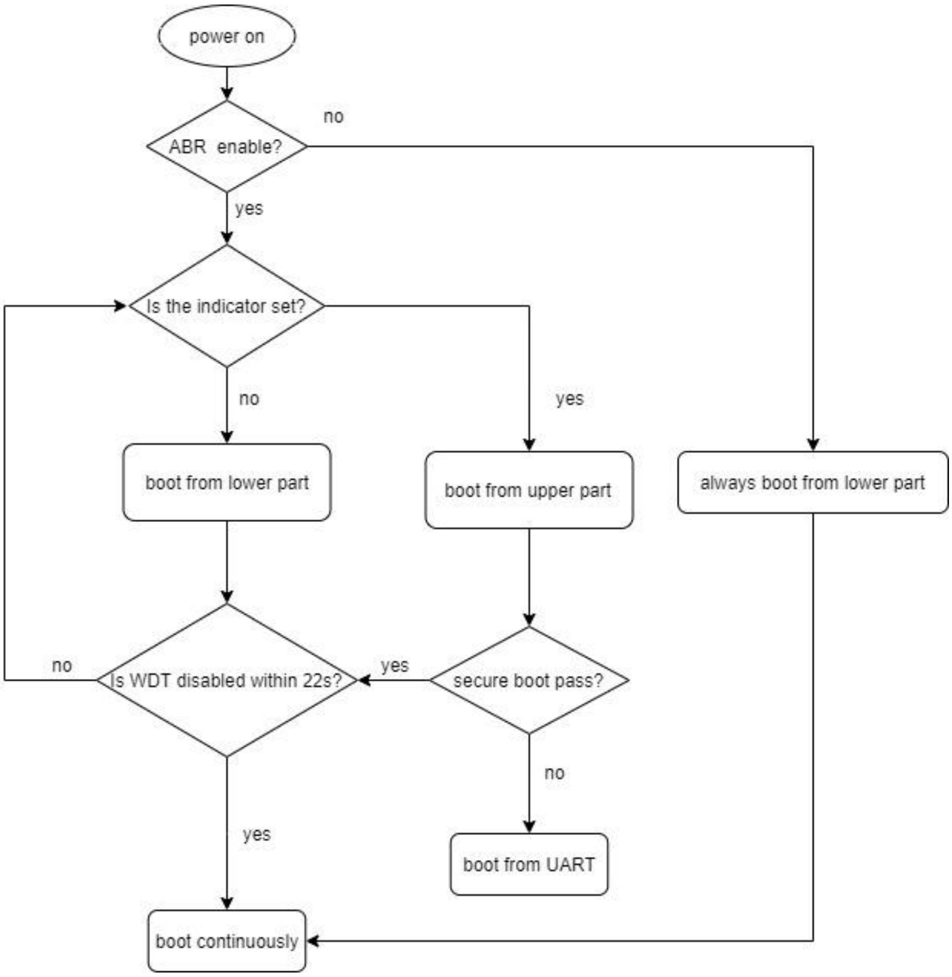
WDT is triggered automatically after power-on when FMC ABR feature is enabled. Indicator is set or cleared whenever WDT timeout. The initial value of indicator is zero and is set after first WDT timeout. Then, BMC uses the upper part to boot up system. If WDT is still not disabled, indicator will be cleared after second WDT timeout and boot part will be changed from the upper part to the lower part. Thus, please keep in mind, if the device boots up successfully, WDT **should be disabled by FW within 22s.** Table abr-t-4 outlines the characteristics of WDT and the indicator for AST1030 platform.

	AST1030
WDT name / control register	FMC_WDT2 / 0x7e620064
Time out period after reset	22 second (default)
How to disable WDT?	clear 0x7e620064[0]
ABR indicator location	0x7e620064[4]
How to clear indicator?	write 0xea to 0x7e620064[23:16]

▲Table abr-t-4

When secure boot is enabled on AST1030-A1 (or after) and the following two conditions are matched, device will trigger boot from UART mechanism, as figure abr-f-10. Be notice, boot from UART external strap is not needed for this scenario.

- MCU ROM fails to verify image on SRAM.
- FMC ABR indicator is set.



▲Figure abr-f-10

8.3. FW Update with FMC ABR Enabled

In this section, an important topic, how to update FW when FMC ABR is enabled, is depicted. The major purpose of FMC ABR feature is to update or recovery original boot image by the other one. If the original boot image is crashed and cannot be used to boot up BMC anymore, the boot image stored in the other flash (part) can be utilized to rescue this trouble. On the other hand, someone may want to update both boot images, the primary one and the alternate one, due to some FW security defects.

For FMC two flashes ABR mode, if the system is boot from the alternate flash, **please clear FMC ABR indicator before implementing FW update process**. When FMC ABR indicator is cleared, the flash address mapping is the same as the scenario where ABR feature is disabled.

For FMC single flash ABR mode, if BMC is booted from the alternate part (upper), please **clear FMC ABR indicator before implementing FW update process**. When FMC ABR indicator is cleared, the flash address mapping is the same as the scenario where ABR feature is disabled.

It is convenient to clear indicator first because of:

- Easy to understand:

  User just needs to care about which flash or part is the updated target and do not need to know which flash or part is used to boot up BMC system currently.

- IO modes, user-mode and normal-write mode, compatible:

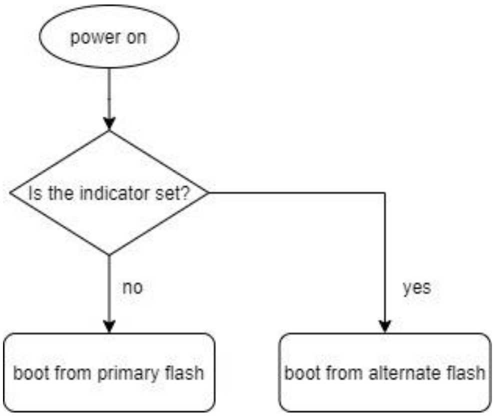
  If user uses SPI user-mode to update FW, the address passed to the flash is the physical address offset of that flash. The address here is the raw address. In contrast, if the normal-write mode is adopted to update flash, the address is modified/mapped by controller before sending to physical flash. It is difficult to explain/execute the update process for FMC single flash ABR mode.

8.4. SPI1 ABR with Two Flashes

AST1030 SPI1 (Host SPI) supports ABR feature for host site. Host can select which flash (part) used to boot up system. Similar to FMC ABR, there are also two cases for SPI1 ABR, two flashes (ABR mode = 0) and single flash (ABR mode = 1). Table abr-t-5 lists configurations for enabling SPI1 two flashes ABR.

	OTPSTRAP[48] = 1 (trap_en_hspiabr)	OTPSTRAP[48] = 0 (trap_en_hspiabr)
HW configuration	OTPSTRAP[50] = 0 (trap_hspi_abrmode)	SPIR64[6] = 0
	OTPSTRAP[53:51] (trap_hspi_size) (optional) - 3b’000: flash size is configured by FW (SPIR64[3:1]) - 3b’001: 2MB - 3b’010: 4MB - 3b’011: 8MB - 3b’100: 16MB - 3b’101: 32MB - 3b’110: 64MB - 3b’111: 128MB	

▲Table abr-t-5



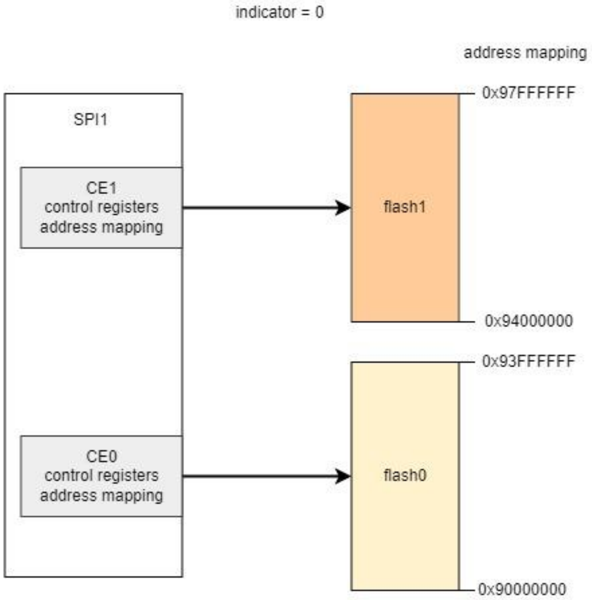
▲fig. abr-f-11

Figure abr-f-11 shows the boot flash selection mechanism for SPI1. Obviously, the indicator, SPIR64[4], decides which boot source is used. The default value of SPIR64[4] is zero and can be configured by the rules in table abr-t-6.

	OTPSTRAP[48] = 0 or 1
How to set indicator?	writes “1” to SPIR64[4]
How to clear indicator?	write “0xea” to SPIR64[23:16]

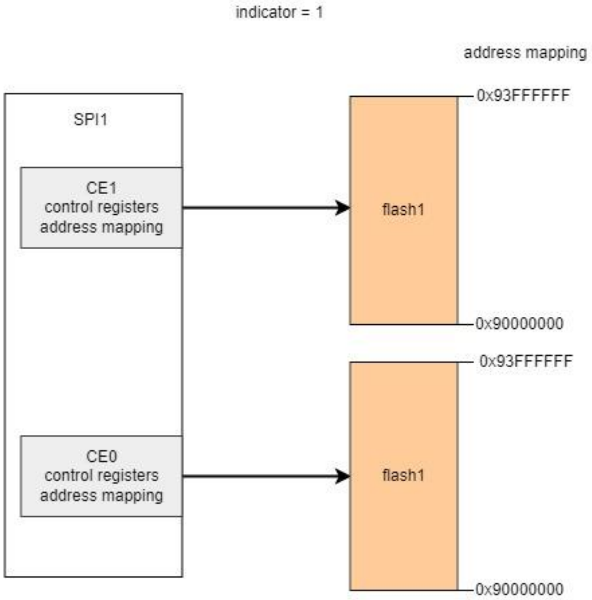
▲Table abr-t-6

When SPI1 two flashes ABR is enabled and SPI1 ABR indicator is zero, the address mapping for SPI1 CE0 and SPI1 CE1 is controlled by SPIR30 and SPIR34. If both CE0 flash (flash0) and CE1 flash (flash1) size is 64MB, the initial address mapping of both flashes is shown as fig. abr-f-12.



▲Figure abr-f-12

The address mapping of flash0 starts from beginning of SPI1 address mapping, namely, 0x90000000 and the address mapping of flash1, which can be configured by SPIR34, is concatenated after flash0 in this example. However, when the indicator is set, the situation is changed, as fig. abr-f-13.



▲Figure abr-f-13

From the SPI1 controller’s view, now, the address mapping for both SPI1 CE0 and SPI1 CE1 are the first 64MB from 0x90000000 and the content for each mapped area is flash1 (alternate flash). Under this situation, flash0 (primary flash) cannot be accessed by SPI1 controller.

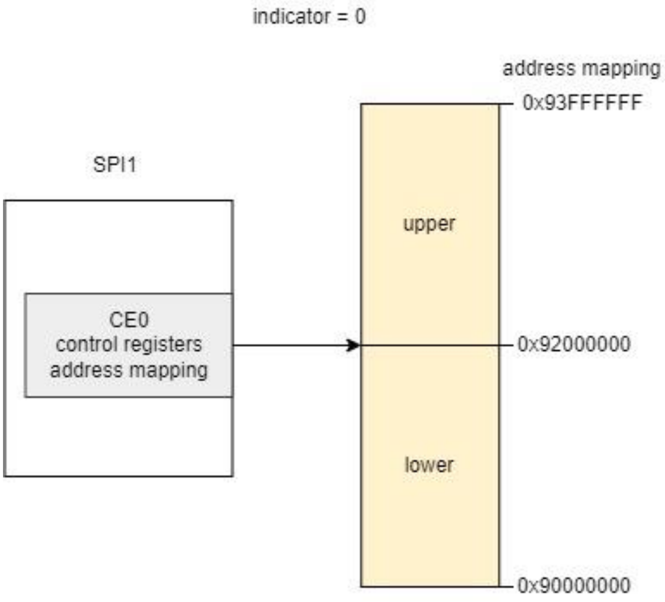
8.5. SPI1 ABR with Single Flash

SPI1 ABR with single flash mode sometimes appears on some specific products, which concept is like SPI1 two flashes ABR mode. Table abr-t-7 lists the HW settings for SPI1 single flash ABR mode (ABR mode = 1).

	OTPSTRAP[48] = 1 (trap_en_hspiabr)	OTPSTRAP[48] = 0 (trap_en_hspiabr)
HW configuration	OTPSTRAP[50] = 1(trap_hspi_abrmode)	SPIR64[6] = 1
	OTPSTRAP[53:51] (trap_hspi_size) (**mandatory**) - 3b’000: flash size is configured by FW (SPIR64[3:1]) - 3b’001: 2MB - 3b’010: 4MB - 3b’011: 8MB - 3b’100: 16MB - 3b’101: 32MB - 3b’110: 64MB - 3b’111: 128MB	

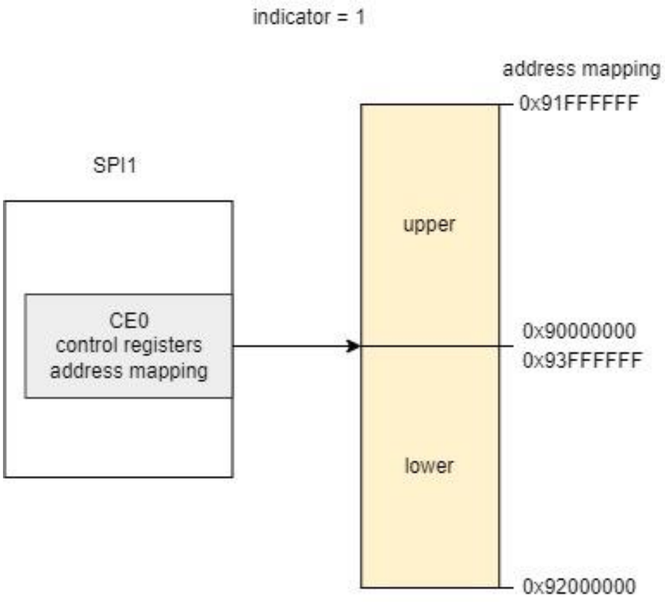
▲Table abr-t-7

SPI1 single flash ABR, as its name implies, is implemented independently by CE0 only. A flash is divided into two equal parts. For convenience, we label “upper” and “lower” for these two parts, refer to fig. abr-f-14 (flash size is 64MB). The size of each part is the half of total flash size. As a result, total flash size must be determined by OTPSTRAP[53:51] or by configuring S0PIR64[3:1] in advance. **Notice, if flash size setting of OTPSTRAP[53:51] is larger than or equal to 16MB, flash itself should be 4B address mode by default since SPI controller is 4B by default under this condition.**



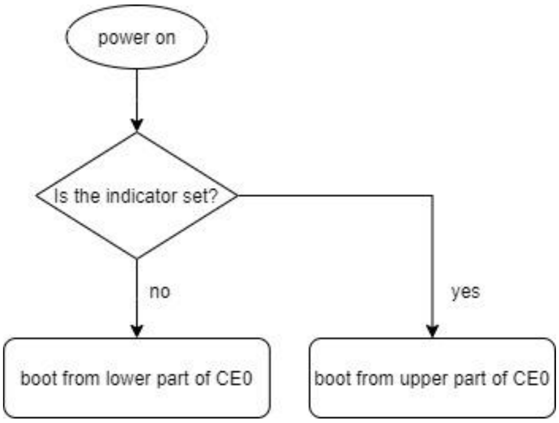
▲Figure abr-f-14

The initial value of SPI1 ABR indicator is zero. When it is set, the address mapping for physical flash upper part and lower part is swapped, shown in fig. abr-f-15.



▲Figure abr-f-15

The SPI1 single flash ABR mode's behavior can be summarized in fig. abr-f-16.



▲Figure abr-f-16

According to fig. abr-f-16, SPI1 ABR indicator determines which part of CE0 is adopted. When indicator is set, upper part is used, otherwise, lower part. Table abr-t-8 outlines how the SPI1 ABR indicator is controlled.

	OTPSTRAP[48] = 0 or 1
	OTPSTRAP[50] = 1 or SPIR64[6] = 1

How to set indicator?	writes “1” to SPIR64[4]
How to clear indicator?	write “0xea” to SPIR64[23:16]

▲Table abr-t-8

8.6. Host Image Update with SPI1 ABR Enabled

Image update is an important feature for a stable and secure system. In this section, we will describe how to update host image by BMC uboot command and some pseudo steps are listed for reference. The similar sequences can be adopted if someone wants to update SPI1 flash0/flash1 image from the host site through SPI1 controller when SPI1 ABR is enabled.

For SPI1 two flashes ABR mode, if the host is boot from the alternate flash, **please clear SPI1 ABR indicator before implementing image update process**. When SPI1 ABR indicator is cleared, the flash address mapping is the same as the scenario where ABR feature is disabled.

For SPI1 single flash ABR mode, if the host is booted from the alternate part (upper), please **clear SPI1 ABR indicator before implementing FW update process**. When SPI1 ABR indicator is cleared, the flash address mapping is the same as the scenario where ABR feature is disabled.

Reader can easily learn that clearing SPI1 ABR indicator is the main principle because of:

- Easy to understand:

  User just needs to care about which flash or part is the updated target and do not need to know the current address mapping for CE0 and CE1. Besides, if SPI1 two flashes ABR mode is enabled and the related indicator is set, SPI1 controller cannot access flash0 (primary flash) and thus, the content of flash0 cannot be updated under this condition.

- IO modes, user-mode and normal-write mode, compatible:

  If SPI user-mode is utilized to update image, the address passed to the flash is the physical address offset of that flash. The address here is the raw address. In contrast, if the normal-write mode is adopted to update flash, the address is modified/mapped by controller before sending to physical flash. It is difficult to explain/execute the update process for SPI1 single flash ABR mode.

9. One Time Programmable Memory

AST1030 built-in 64Kbit one time programmable (OTP) memory for configuration, strap, key storage, patch and user data. Each memory bit cell inside the OTP memory is capable to be programmed once.

Typically, the data stored the OTP memory are non-volatile and can preserve permanently, but to improve the FIT (failure in time) of the OTP memory, ECC is recommended to enable.

9.1. OTP Memory Organization

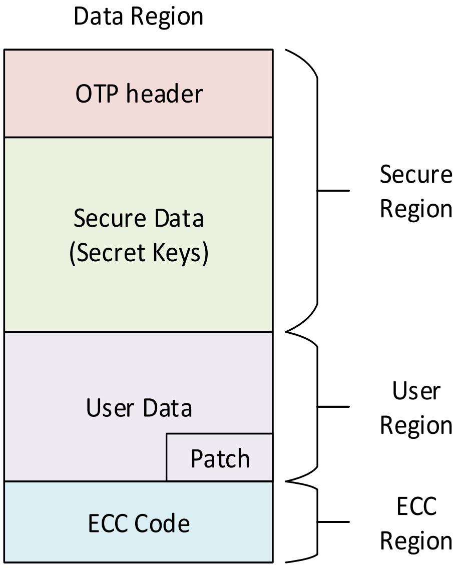
The OTP memory comprises three physical regions. Each region has its own addressing space. The OTP memory regions are:

- Configuration region: Total 1k bits.
- Data region: Total 64k bits.
- Redundancy region: Total 2k bits to repair at most 4 defects.

Configuration Region
Data Region
Redundancy Region

▲Figure OTP hardware architecture

9.1.1. OTP Data Region



▲Figure OTP data region layout

AST2600 OTP data region is divided into a few logic regions

- OTP Header: defines key type and location
- Secure Region: for OTP header and secure data (keys).
- User Region: use defined.
- OTP patch: reserved 32 DW for OTP patch at the end of user data region.
- ECC Region: Optional ECC for both secure and user regions.

9.1.2. OTP Header Format

The Secure Region includes the OTP Header and Secure Data.

Here is the OTP Header Data Structure:

	OTP Header
Bit	Description
31:20	**Key Length for RSA exponent bit length.**
19:18	**Key Length** When the [17] = 1, it's RSA modulus bit length. 00: RSA1024 01: RSA2048 10: RSA3072 11: RSA4096
17:14	**Key Type** 0000: Empty Header 0001: AES-256 as secret vault key 0010: AES-256 as OEM platform key for image encryption/decryption in Mode 0101: ECDSA384 domain parameters (big endian), If the SBE cannot find this key, the FIPS 183-3 curve P384 will be used as default. 0111: ECDSA384 (big endian) as OEM DSS public keys 1000: RSA-public (little endian) as OEM DSS public keys 1010: RSA-public (little endian) as SOC key 1110: RSA-private (little endian) as SOC key 1001: RSA-public (big endian) as OEM DSS public keys 1011: RSA-public (big endian) as SOC key 1100: RSA-private (little endian) as SOC key 1101: RSA-private (big endian) as SOC key 1111: Empty Header Others: Reserved (Note: The ECDSA is not supported in A0) (Note: The little endian is supported only in A0)
13	**Last List** When set, it indicates that this is the last list of the data
12:3	**Key Offset (8-byte aligned)**
2:0	**Key Number ID** For key type OEM DSS public keys in Mode 2 and AES-256 in Mode GCM only. Number ID 0 is for low security key. For other type of keys, value is 0.

9.1.3. Secure Key Data Structure

	RSA 1024
Byte Range	Description
000 - 07F	Modulus
080 - 0FF	Exponent (private key)

	RSA 2048
Byte Range	Description
000 - 0FF	Modulus
100 - 1FF	Exponent (private key)

	RSA 3072
Byte Range	Description
000 - 07F	Modulus
080 - 0FF	Exponent (private key)

	RSA 4096
Byte Range	Description
000 - 17F	Modulus
180 - 2FF	Exponent (private key)

	AES 256 (OEM platform key)
Byte Range	Description
000 - 01F	AES Key Byte 0 ~ 31

	AES 256 (Secret Vault Key)
Byte Range	Description
000 - 01F	First AES Key Byte 0 ~ 31
020 - 03F	Second AES Key Byte 0 ~ 31

9.2. OTP tool

Using this tool to generate the otp image, and using OTP Utility to program that image into OTP memory.

❖ **Location**

GIT: https://github.com/AspeedTech-BMC/socsec
It includes both socsec and otp tools.
Sample Config Folder:
boards/arm/ast1030_evb/otp_config/
Sample Key Folder:
boards/arm/ast1030_evb/key/

❖ **Usage**

The following is OTP image generating command.

```
# otptool make_otp_image --help

usage: otptool make_otp_image [-h]
       [--key_folder KEY_FOLDER]
       [--user_data_folder USER_DATA_FOLDER]
       [--output_folder OUTPUT_FOLDER]
       [--no_last_bit] [--no_pre_production]
       config

positional arguments:
  config                configuration file

optional arguments:
  -h, --help            show this help message and exit
  --key_folder KEY_FOLDER
                        key folder
  --user_data_folder USER_DATA_FOLDER
                        user data folder
  --output_folder OUTPUT_FOLDER
                        output folder
  --no_last_bit         (develop)remove last bit in OTP header
  --no_pre_production   check no pre production version
```

Argument:

- config: the config file is a json format document, which content otp data region, otp config region and otp strap description. Below is an example.
- key_folder: put all key file into key folder
- user_data_folder: put all user data file into key folder
- output_folder: the generated otp image will put into this folder.

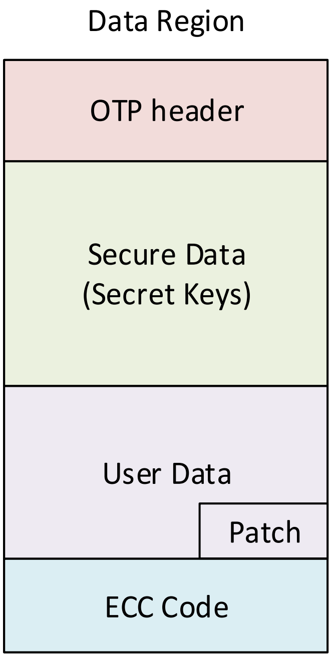
- no_last_bit: remove last bit in OTP header.
- no_pre_production: It helps to verify whether users set the correct SOC version in their providing OTP config.

Output:

- otp-all.image: a programmable image, use uart otp utility to program this image, which contain all region of otp.
- *.image: a programmable image, use rt otp utility to program this image
- otp-data.bin: raw data of otp data region

❖ OTP data region:

1. OTP Header: 16 DWs (512 bits) (default)
2. Secure Data: 1024 DW (32768 bits) (user defined by OTPCFG0[21:16]
3. User data: 752 DW (24064 bits)
4. OTP patch: reserved 32 DW for otp patch at the end of user data region.
5. ECC data: 256 DW (8192 bits) (when ECC is enable)



▲Figure data region

"config_region" and "otp_strap" object is to describe the otp config region. The default values of OTP configuration memories are all 0s. They can be programmed to 1s only and can't programmed back to 0s again. All OTP memory can only be programmed once, only OTP strap can be updated 6 times.

Using the following command to check the remaining times for update.

```
ast# otp info strap v
BIT(hex) Value  Remains  Reg_Protect Protect   Description
__________________________________________________________________________________
0x0     0x0    6       0x0        0x0       Disable secure boot
```

The following is OTP image printing command.

```
# otptool print --help
usage: otptool print [-h] otp_image

positional arguments:
  otp_image   OTP image

optional arguments:
  -h, --help  show this help message and exit
```

Argument:

- The path of otp image.

Output

- Print the detail information of input otp image.

❖ **Example**

```
# otptool make_otp_image \
./boards/arm/ast1030_evb/otp_config/1030A0_RSA2048_SHA256.json \
--key_folder ./boards/arm/ast1030_evb/key \
--output_folder ./RSA2048_SHA256
```

9.3. OTP Utility

There are two OTP utility to program or print the OTP, Zephyr OTP utility and UART OTP Utility, the first is Zephyr command, and the second is run through the boot from uart. Zephyr otp utility do not support programming otp image currently.

9.3.1. Zephyr OTP Utility

Under Zephyr console, the otp command provide to perform read/write access to the One-Time-Programmable memory.

Usage:

```
uart:~$ otp help
otp - ASPEED One-Time-Programmable sub-system
      otp version
      otp read conf|data <otp_dw_offset> <dw_count>
      otp read strap <strap_bit_offset> <bit_count>
      otp info strap [v]
      otp info conf [otp_dw_offset]
      otp info scu
      otp info key
      otp pb conf|data [o] <otp_dw_offset> <bit_offset> <value>
      otp pb strap [o] <bit_offset> <value>
      otp protect [o] <bit_offset>
      otp scuprotect [o] <scu_offset> <bit_offset>
      otp update [o] <revision_id>
      otp retire [o] <key_id>
      otp rid
```

Example:

Read OTP data, starting from 0 and read 0x20 double words.

```
uart:~$ otp read data 0 20
000: 00000000 FFFFFFFF 00000000 FFFFFFFF
010: 00000000 FFFFFFFF 00000000 FFFFFFFF
020: 00000000 FFFFFFFF 00000000 FFFFFFFF
030: 00000000 FFFFFFFF 00000000 FFFFFFFF
040: 00000000 FFFFFFFF 00000000 FFFFFFFF
050: 00000000 FFFFFFFF 00000000 FFFFFFFF
060: 00000000 FFFFFFFF 00000000 FFFFFFFF
070: 00000000 FFFFFFFF 00000000 FFFFFFFF
```

Print otp configuration information.

```
uart:~$ otp info conf
DW    BIT        Value      Description
__________________________________________________________________________
0x0   0x0        0x0        Enable Secure Region programming
0x0   0x1        0x0        Disable Secure Boot
…
```

Program one bit into OTP strap.

```
uart:~$ otp pb strap 0 1
BIT(hex)  Value  Option          Status
___________________________________________________________________________
0x0       0      0 0 0 0 0 0 0    not protected and still can write 7 times
OTPSTRAP[0]:
    This bit will be protected and become non-writable.
    Write 1 to OTPSTRAP[0] OPTION[1], that value becomes from 0 to 1.
type "YES" (no quotes) to continue:
```

Program one bit into OTP configuration region.

```
uart:~$ otp pb conf 0 7 1
Program OTPCFG0[7] to 1
type "YES" (no quotes) to continue:
```

Protect one bit of OTP strap.

```
uart:~$ otp protect 5
OTPSTRAP[5] will be protected
type "YES" (no quotes) to continue:
```

Print OTP revision ID

```
uart:~$ otp rid
current SW revision ID: 0x1
current OTP revision ID: 0x0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
___________________________________________________
 0 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
10 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
20 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
30 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Update OTP revision ID

```
uart:~$ otp update 1
current OTP revision ID: 0x0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
___________________________________________________
 0 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
10 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
20 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
30 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
input update number: 0x1
OTPCFGA[0] will be programmed
type "YES" (no quotes) to continue:
YES
OTP revision ID: 0x1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
___________________________________________________
 0 | 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
10 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
20 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
30 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
SUCCESS
```

9.3.2. UART OTP Utiliy

Please refer to the UART OTP Utility user guide for detail.

10. Secure Boot

AST1030 support two root of trust (RoT) measurement mode: Mode_2 and Mode_ecdsa.

10.1. Root of Trust (RoT)

AST1030 support hardware boot image measurement and decryption, the verification key or encryption key will put in OTP. RoT boot image can storage in SPI and the boot image size is limited to 768KB. When secure boot failed, the MCU will automatically switch to boot from uart mode

1.2.1 Image Encryption/Decryption

1. Support only when OTPCFG0[26] and OTPCFG0[27] are set. The image inside the SRAM will be decrypted. Mode 2 can support image encryption.
2. The image encryption algorithm is AES-256. The encryption key is OEM platform key can be:
   i. In OTP memory: the key type is "AES 256 OEM platform key". This is default mode.
   ii. In SPI flash: The OEM platform key is encrypted key and Secure Boot Header DW 0x0 indicates the location. The OEM platform key shall be fetched and decrypted by OEM private key or OEM public key in OTP.

1.2.2 Multiple Secure Keys

1. Support one low security key and multiple high security keys.
2. Low security key can be used during firmware development before production. The low security key is selectable by hardware Strap Option which can be ignored once in production by permanently disabling by set OTPCFG0-D[5].
3. Secure Boot engine will check security keys from key #1. When the key is disabled or the signature check fail, use the next key and check again until all programmed keys are checked
4. When the firmware boots with a higher number key then the disabling of lower number keys are allowed. For example, when # N key is compromised:
   i. Sign the firmware by using #N+1 key if the #N+1 key is not compromised.
   ii. Update firmware to SPI flash.

iii. Boot firmware with #N+1 key signed.

iv. Disable #N key.

Otherwise, the key #N is protected by hardware and cannot be disabled.

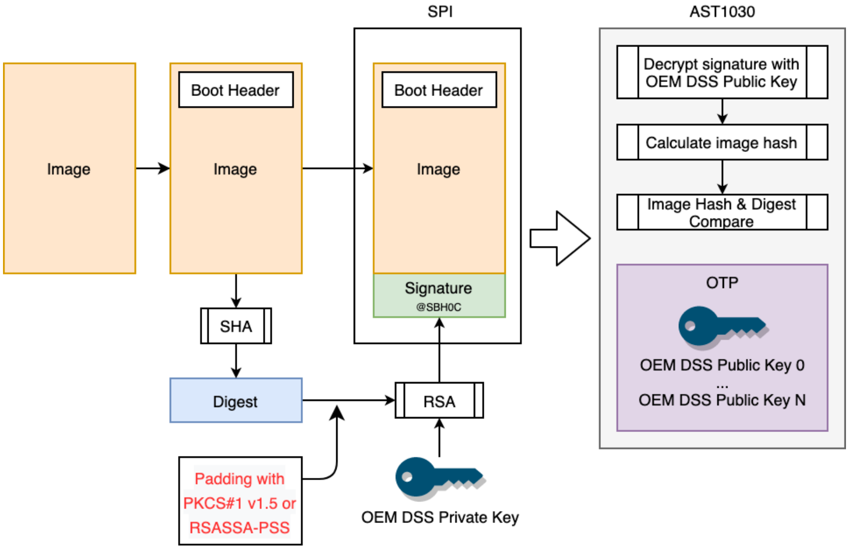
1.2.3 RoT Boot Header

The Secure Boot header is 8 double words of data in SPI flash (or eMMC). By default is located at 0x20 (The offset is configurable in the OTP configuration region "OTPCFG2")

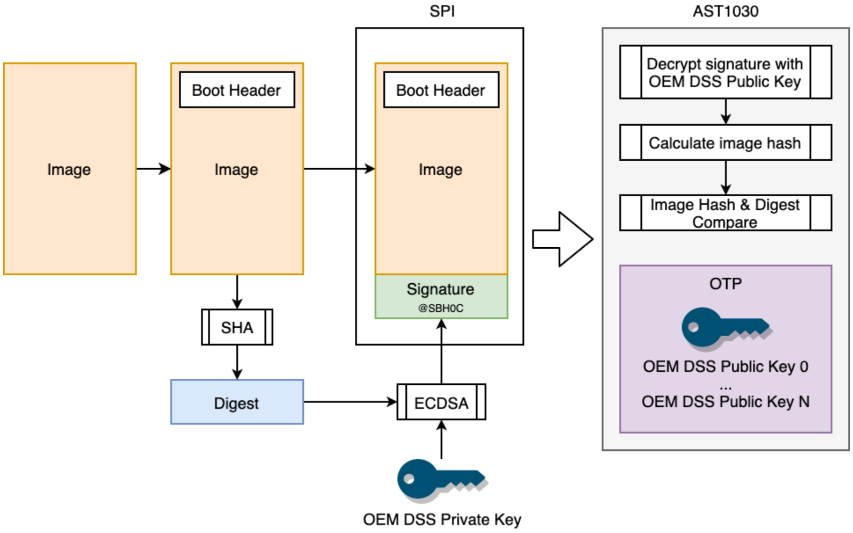
Name	Offset	Description	Note
SBH00	0x0	Key location	@+0: OEM platform key or IV location
SBH04	0x4	Start address of encrypted image	
SBH08	0x8	Image size	Must be 16KB (A0) Must be multiple of 512 byte. Minimum size is 16KB. Maximum size is 768KB.
SBH0C	0xC	Signature location	@+0: Signature 512 bytes
SBH10	0x10	Header revision ID[31:0]	Header revision ID[63:0] must be equal to or greater than Manifest ID (OTPCFG10 and OTPCFG11) for rollback prevention.
SBH14	0x14	Header revision ID[63:32]	
SBH18	0x18	Flash patch code location	0: means no flash patch code Maximum value is 768KB.
SBH1C	0x1C	Checksum	Sum of all 8 DWs must be 0

10.2. RoT Boot Mode

10.2.1. Mode 2 and Mode ECDSA without Firmware Encryption



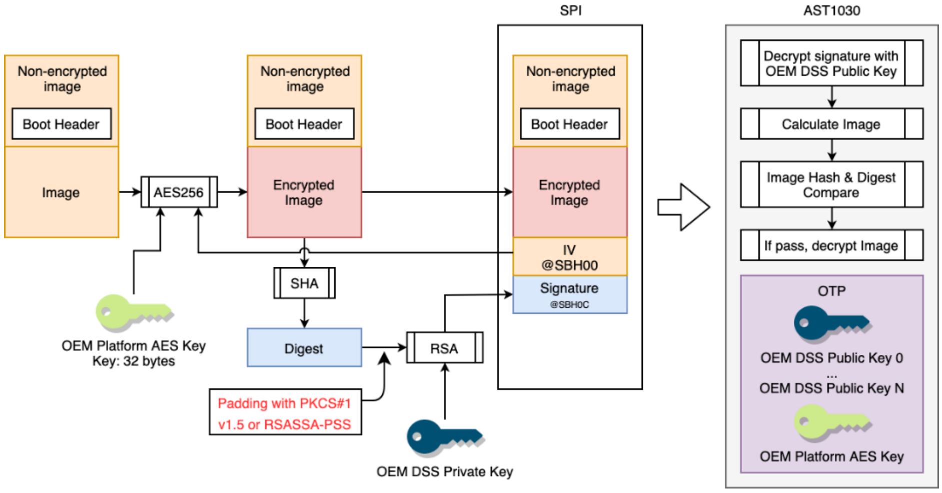
▲Figure Mode 2 boot procedure



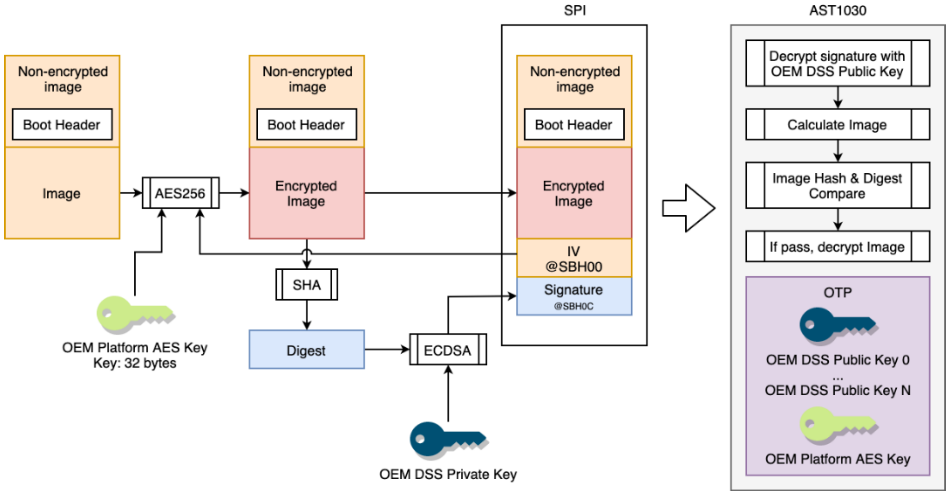
▲Figure Mode ECDSA boot procedure

The measured value and the resultant "Signature" is encrypted with OEM DSS Private Key and the resultant encrypted value is stored with the firmware Image in the Flash device. The OEM DSS Public Key is programmed by the ODM/OEM to the OTP memory which is used by the Secure Boot hardware to measure and compare the firmware.

10.2.2. Mode 2 and Mode ECDSA for encrypted image with OEM platform key (option 1)



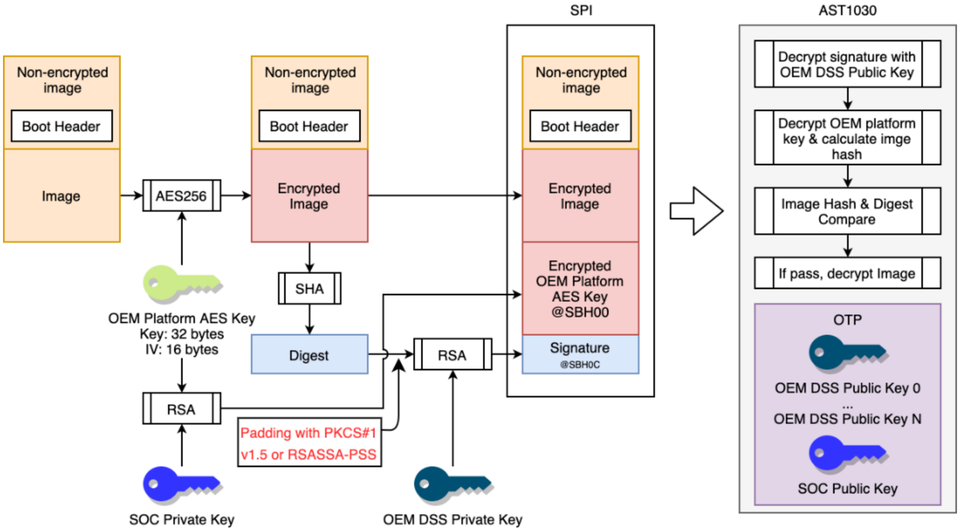
▲Figure Mode 2 encrypted option1 boot procedure



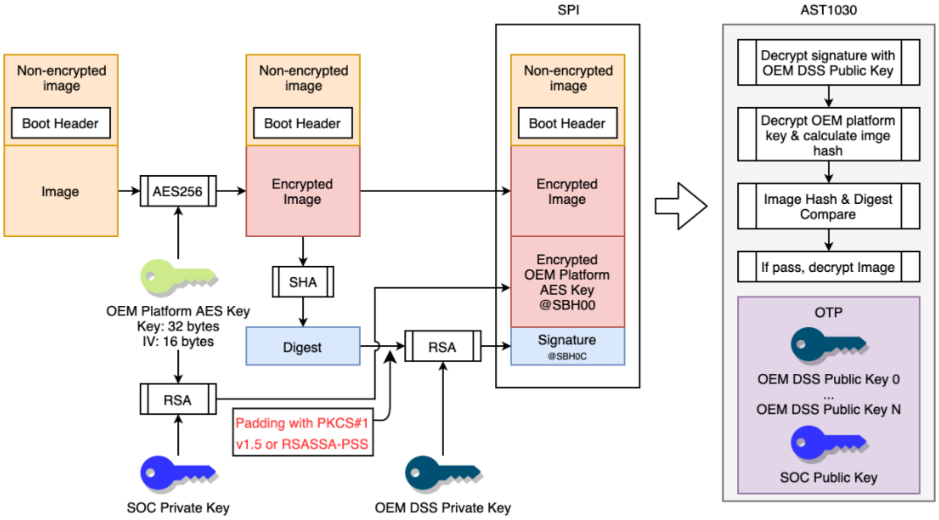
▲Figure Mode ECDSA encrypted option1 boot procedure

The AES Encryption Key, which is named as “OEM Platform key”, used for image encryption is default programmed to the OTP memory. The initial vector “IV” used for image encryption is separately stored in the Flash Device. The measured value of “Encrypted image” is separately encrypted with the OEM DSS Private Key and the resultant “Signature” is also stored in the Flash Device. The ODM/OEM can program the OEM DSS Public Keys and OEM Platform key in the OTP memory at the time of manufacturing. The OTP programmed keys are used to extract the other keys, measure the firmware and compare with the programmed “Signature”. When secure boot controller find “OEM Platform key” in OTP memory, the option 1 will be enabled.

10.2.3. Mode 2 and Mode ECDSA for encrypted image with SOC key (option 2)



▲Figure Mode 2 encrypted option2 boot procedure



▲Figure Mode ECDSA encrypted option2 boot procedure

The AES Encryption Key which is named as “OEM Platform key” associated with IV (initial vector) used for image encryption is separately encrypted with the SOC Private Key (optional SOC Public Key) and the resultant “Encrypted OEM Platform Key” is stored along with the Encrypted Firmware Image on the Flash device. The measured value of “Encrypted image” is separately encrypted with the OEM DSS Private Key and the resultant “Signature” is also stored in the Flash Device. The ODM/OEM can program the OEM DSS Public Keys and SOC key in the OTP memory at

the time of manufacturing. The OTP programmed keys are used to extract the other keys, measure the firmware and compare with the programmed “Signature”.
When secure boot controller find “SOC public key” or “SOC private key” in OTP memory, the option 2 will be enabled.

10.3. SOC Secure Tool

This tool is used to generate ASPEED secure boot RoT image. Please reference the README for more details.

❖ **Location**

GIT: https://github.com/AspeedTech-BMC/socsec
It includes both socsec and otp tools.

❖ **Setting up**

```
# sudo apt-get install python3 python3-pip python3-virtualenv
# virtualenv .venv
# source .venv/bin/activate
# pip3 install -r requirements.txt
# python3 setup.py install
```

10.3.1. Key generation

❖ **RSA Key**

Creating an RSA key pair and AES key for RoT and OTP images.
RSA key for secure image generation tools, size 2048 bits

```
# openssl genrsa -out rsa_key.pem 2048
```

The generated rsa_key.pem contain both public and private key, you can do the following command to generate rsa key, which only contain public key.

```
# openssl rsa -in rsa_key.pem -pubout -out rsa_key-public.pem
```

❖ **ECDSA key**

Generate ECDSA private key.

```
# openssl ecparam -name secp384r1 -genkey -out ecdsa_key.pem
```

Generate ECDSA public key from private key.

```
# openssl ec -in ecdsa_key.pem -pubout -out ecdsa_key-public.pem
```

❖ **AES key**

Using the following command to generate the AES key randomly

```
# openssl rand 32 > aes_key.bin
```

10.3.2. Using SOCSEC

The following is RoT image generating command.

```
# socsec make_secure_bl1_image -h

usage: socsec make_secure_bl1_image [-h]
       [--soc SOC]
       [--bl1_image BL1_IMAGE]
       [--stack_intersects_verification_region {true,false}]
       [--header_offset HEADER_OFFSET]
       [--rsa_sign_key RSA_SIGN_KEY]
       [--rsa_key_order ORDER]
       [--gcm_aes_key GCM_AES_KEY]
       [--output OUTPUT]
       [--algorithm ALGORITHM]
       [--rollback_index ROLLBACK_INDEX]
       [--signing_helper [APP]]
       [--signing_helper_with_files [APP]]
       [--enc_offset ENC_OFFSET]
       [--aes_key [AES_KEY]]
       [--key_in_otp]
       [--rsa_aes [RSA_AES]]
       [--flash_patch_offset FLASH_PATCH_OFFSET]
       [--cot_algorithm [ALGORITHM]]
       [--cot_verify_key [COT_VERIFY_KEY] | --cot_digest COT_DIGEST]

optional arguments:
  -h, --help           show this help message and exit
  --soc SOC            soc id (e.g. 2600, 1030)
  --bl1_image BL1_IMAGE
                      Bootloader 1 Image (e.g. u-boot-spl.bin), which
will
                      be verified by soc
  --stack_intersects_verification_region {true,false}
                      By default, the maximum size of SPL images socsec
                      will sign is 60KB, since, historically, the SoCs
                      have been using the top of the SRAM for the SPL
                      execution stack. However, on 2600 (A1) and above
                      SoCs, an
                      additional 24KB SRAM can be used for the stack,
                      allowing the verification region to occuppy the
                      entire 64KB (including signature). For these models
                      of boards, this layout will also be the default in
                      future SDK
                      releases. Use this parameter to explicitly indicate
                      that the SPL image being signed has (=true) or has
                      not (=false) the SPL stack overlapping the 64KB
                      verification region. With this argument set to
                      'false',
                      socsec will sign SPL images up towards 64KB
(including 512B signature)
  --header_offset HEADER_OFFSET
                      RoT header offsest
  --rsa_sign_key RSA_SIGN_KEY
                      Path to RSA private key file, which will use to
```

```
sign BL1_IMAGE
  --rsa_key_order ORDER
                      This value the OTP setting(e.g. little, big),
                      default value is "little"
  --gcm_aes_key GCM_AES_KEY
                      Path to aes private key file, which will use to
sign BL1_IMAGE
  --output OUTPUT     Output file name
  --algorithm ALGORITHM
                      Algorithm to use (default: NONE e.g. AES_GCM,
                      AES_RSA2048_SHA256, RSA2048_SHA256, ...), RSA algo
                      support RSA1024, RSA2048, RSA3072 and RSA4096, HASH
                      algo support SHA224, SHA256, SHA384 and SHA512
  --rollback_index ROLLBACK_INDEX
                      Rollback Index
  --signing_helper [APP]
                      Path to helper used for signing
  --signing_helper_with_files [APP]
                      Path to helper used for signing using files
  --flash_patch_offset FLASH_PATCH_OFFSET
                      Flash patch offset for ast2605

enc_group:
  Enable aes encryption in mode 2

  --enc_offset ENC_OFFSET
                       Offset where encryption start
  --aes_key [AES_KEY]  Path to aes key file
  --key_in_otp         aes key is storing in otp
  --rsa_aes [RSA_AES]  Path to RSA public key file, which is used to
encrypt aes key
```

❖ Example

```
# socsec make_secure_bl1_image \
--soc 1030 \
--algorithm RSA2048_SHA256 \
--bl1_image ./build/zephyr/zephyr.bin \
--output ./build/zephyr/s_zephyr.bin \
--rsa_sign_key
/boards/arm/ast1030_evb/key/test_oem_dss_private_key_2048_1.pem
```

11. Reference

[1] "zephyrproject/zephyr-rtos," [Online]. Available: https://github.com/zephyrproject-rtos/zephyr.

[2] "Aspeed Zephyr Kernel tag v00.01.05," Aspeed Technology, [Online]. Available: https://github.com/AspeedTech-BMC/zephyr/tree/v00.01.05.

[3] "Zephyr Project Documentation," [Online]. Available: https://docs.zephyrproject.org/latest/.

[4] "zephyr-rtos tag zephyr-v2.6.0," [Online]. Available: https://github.com/zephyrproject-rtos/zephyr/tree/zephyr-v2.6.0.