TX2 MB1 Platform Configuration

In the boot sequence for Jetson TX2, MB1 uses the MB1 BCT to configure platform-specific static settings. MB1 executes before any other CPUs are enabled. The MB1 stage is owned by NVIDIA and signed by NVIDIA and the OEM.

For more information, see TX2 Boot Flow.

About MB1 BCT

MB1 BCT specifies platform-specific data. When Tegraflash is invoked to flash a platform, it calls the `tegrabct_v2` tool to create the MB1 BCT. It uses the following data:

- Platform configuration files
- `tegrabl_mb1_bct.h` header file

The BCT required by this stage is signed by the OEM. The MB1 stage offers to perform platform specific initialization. It also sets up the secure control register (SCR).

Platform-specific configuration files specify:

- Pinmux and GPIOs configuration
- Prod setting
- Pad voltage setting
- PMIC setting
- Secure register configuration

These configuration files are at:

```
<top>/bootloader/<platform|ver>/BCT/
```

TX2 Pinmux and GPIO Configuration

The Jetson TX2 pinmux configuration file provides pinmux and GPIO configuration information. The typical format for this data is register address and data, as a pair. MB1 only allows writes to the pinmux and GPIO address range, from this table.

The pinmux configuration file is at:

```
<top>/bootloader/<platform|ver>/BCT/tegra<t-arch>-mb1-bct-pinmux-gp
io-<board>-<board_revision>.cfg
```

Usage

```
pinmux.<address> = <value>;
```

Where:

- `pinmux` is the domain name for GPIO and pinmux configuration data.
- `<address>` is the absolute register address.
- `<value>` is the 32-bit data value.

Device-side implementation

```
write(value, address);
```

Example

```
#### Pinmux for used pins ####

pinmux.0x02434060 = <value1>; # gen1_i2c_scl_pc5.PADCTL_CONN_GEN1_I
2C_SCL_0

pinmux.0x02434064 = <value2>; # gen1_i2c_scl_pc5.PADCTL_CONN_CFG2TM
C_GEN1_I2C_SCL_0

pinmux.0x02434068 = <value1>; # gen1_i2c_sda_pc6.PADCTL_CONN_GEN1_I
2C_SDA_0

pinmux.0x0243406C = <value2>; # gen1_i2c_sda_pc6.PADCTL_CONN_CFG2TM
C_GEN1_I2C_DA_0

                           ::::

#### Pinmux for unused pins for low-power configuration ####

pinmux.0x02434040 = <value1>; # gpio_wan4_ph0.PADCTL_CONN_GPIO_WAN4
_0

pinmux.0x02434044 = <value2>; # gpio_wan4_ph0.PADCTL_CONN_CFG2TMC_G
PIO_WAN4_0

pinmux.0x02434048 = <value1>; # gpio_wan3_ph1.PADCTL_CONN_GPIO_WAN3
_0

pinmux.0x0243404C = <value2>; # gpio_wan3_ph1.PADCTL_CONN_CFG2TMC_G
PIO_WAN3_0
```

TX2 Prod Configuration

The Jetson TX2 `prod` setting is the configuration of system characterization and interface, and controller settings. This is required for the interface to work reliably in a given platform. The prod setting is set at the controller level, and separately at the pinmux level.

The format of this configuration is a tuple of register `address`, `mask`, and data `value`. MB1 reads data from `address`, modifies it based on `mask` and `value`, and then writes it back to `address`.

The prod configuration file is at:

```
<top>/bootloader/<platform|ver>/BCT/tegra<t-arch>-mb1-bct-prod-<boa
rd>-<board_revision>.cfg
```

Usage

```
prod.<address>.<mask> = <value>;
```

Where:

- `prod` is the domain name prefix for the setting.
- `<address>` is the pad control register address.
- `<mask>` is the mask value (4 bytes, unsigned).
- `<value>` is the data value (4 bytes, unsigned).

Device-side implementation

```
val = read(address)
val = (val & ~mask) | (value & mask);
write(val, address);
```

Example

```
prod.0x02436010.0x00006000 = 0x00002000; # SDMMC4_DAT7, DRV_TYPE: D
RIVE_2X
prod.0x02436014.0x00006000 = 0x00002000; # SDMMC4_DAT6, DRV_TYPE: D
RIVE_2X
prod.0x02436018.0x00006000 = 0x00002000; # SDMMC4_DAT5, DRV_TYPE: D
RIVE_2X
prod.0x0243601c.0x00006000 = 0x00002000; # SDMMC4_DAT4, DRV_TYPE: D
RIVE_2X
```

```
prod.0x02436020.0x00006000 = 0x00002000; # SDMMC4_DAT3, DRV_TYPE: D
RIVE_2X
```

TX2 Pad Voltage Configuration

Tegra Jetson TX2 pins and pads are designed to support multiple voltage levels at a given interface. They can operate at 1.2 volts (V), 1.8 V or 3.3 V. Based on the interface and power tree of a given platform, the software must write to the correct voltage of these pads to enable interface. If pad voltage is higher than the I/O power rail, then the pin does not work on that level. If pad voltage is lower than the I/O power rail, then it can damage the SoC pads. Consequently, configuring the correct pad voltage is required based on the power tree.

The pad configuration file is located at:

```
<top>/bootloader/<platform|ver>/BCT/tegra<t-arch>-mb1-bct-pad-quill
-<board>-<board_revision>.cfg
```

Usage

```
pmc.<address> = <value>;
```

Where:

- `pmc` is the domain name prefix for the setting.
- `<address>` is the absolute register address.
- `<value>` is the 32-bit data value.

Device-side implementation

```
write(value, address);
```

Example

```
pmc.0x0c36003c = 0x0000003e; # PMC_IMPL_E_18V_PWR_0
pmc.0x0c360040 = 0x00000079; # PMC_IMPL_E_33V_PWR_0
```

TX2 PMIC Configuration

During system boot, Jetson TX2 MB1 enables system power rails such as CPU, SRAM, CORE, as well as some system PMIC configurations. The typical configurations are:

- Enabling rails
- Setting rail voltages
- FPS configurations

Enabling and setting of voltages of rails may require:

- I2C command to devices
- MMIO access to Tegra registers, either read-modify-write or write-only
- Delay after the commands

Rail-specific configurations, such as I2C commands, MMIO access, and delays, are platform-specific. The MB1 BCT configuration file must provide configuration information.

The MB1-CFG format supports:

- I2C, Pulse Width Modulation (PWM) commands, and MMIO commands on any sequence.

- Any I2C/PWM controller instance.
- Any 7-bit slave address of the device.
- MMIO commands on read-modify-write format to support read only and Read-modify-write format.
- I2C commands are read-modify-write format to support read only and Read-modify-write format.
- PWM commands are for enabling and configuring the PWM.
- Any amount of delay between commands.
- Write only commands for PWM/I2C/MMIO.
- Any size of device registers address and data size for i2c commands.
- I2c command on the 400KHz.
- The sequence may be:
    - 1 MMIO, 1 I2C
    - 1 I2C, 1 MMIO
    - 2 MMIO, 1 I2C
    - 1 MMIO, 2 I2C

The typical rail/configurations are divided into following PMIC command domains:

- Generic: General PMIC configurations.
- CPU: Command related to CPU rails.
- GPU: Commands related to GPU.
- SRAM: Commands related to SRAM.
- CORE: Commands related to CORE.
- MEM: Commands related to Memory.
- THERMAL: Commands for thermal configurations.
- SHUTDOWN: Commands for shutdown related configurations.

If a configuration is NOT identified for given rail, the command sequence of that rail is not required because MB1 device side code ignores the configuration of that rail.

Each rail is defined with a unique ID to make the parsing and BCT binary easier. The unique IDs are as follows:

Rail Name	**ID**
GENERIC	1
CPU	2
CORE	3
SRAM	4
GPU	5
MEM	6
THERMAL	7
SHUTDOWN	8

The PMIC configuration file is at:

```
<top>/bootloader/<platform|ver>/BCT/tegra<t-arch>-mb1-bct-pmic-quil
l-<board>-<board_revision>.cfg
```

Note	**Revie eac po er rail e ore customi ing t e carrier oard.**

sage

Common parameters:

```
pmic.<parameter> = <value>;
```

Rail-specific parameters:

```
pmic.<rail-names>.<rail-id>.<parameters> = <value>;
```

Where common `<parameters>` is one of the following:

Parameter	**Description**
command-retries-count	The number of allowed command attempts.
wait-before-start-bus-clear-us	Wait timeout, in microseconds before issuing the bus clear command. The wait time is calculated as 1 *n* microseconds where *n* is provided by this parameter.
rail-count	Number of rails in this configuration file that need to be configured.

Example

```
pmic.command-retries-count = <value>;
pmic.wait-before-start-bus-clear-us = <value>;
pmic.rail-count  = <value>;
```

Rail-specific parameters take the following format:

- The rail specific commands are divided into blocks.
- Each rail can have one or more blocks. Each block of given rails are indexed starting from 0.
- Each block contains either MMIO or I2C commands. If both MMIO and I2C commands are required, then commands are broken into multiple blocks.
- If a block contains I2C type of commands, then all commands are sent to the same device. If I2C commands are required for multiple devices, then it must be split into multiple blocks.
- If commands on given blocks are I2C type, then the device address, register address size, and register data size parameters are not required for MMIO commands.
- A given block can contain more than one command, but all commands must be of the same type.
- A delay is provided after each command of a given block. The delay is the same for all commands. If different delay is required, it must be split into multiple blocks.

Rail specific parameters are prefixed by the following:

```
pmic.<rail-name>.<rail-id>
```

Parameter	Description
block-count	Specifies the block count. pmic. rail-name . rail-id .block-count value Where value , for block-count, is the number of command blocks for a given rail.
block	Specifies the block identification parameter. All blocks are indexed, starting from 0. pmic. rail-name . rail-id .block index
type	Specifies the command type, either MMIO (0) or I2C (1).
delay	Specifies the delay, in microseconds, after each command in a given block.
count	Specifies the number of commands in a block.

I2 T pe peci ic Parameters

The I2C type specific parameters are as follows:

Parameter	Description
I2c-controller-id	Controller ID of I2C.
slave-add	7-bit slave address.
reg-data-size	Register size in bits: 0 or 8:1 byte 16: 2 byte
reg-add-size	Register address size in bits: 0 or 8:1 byte 16: 2 byte

ommands

Commands can be either MMIO or I2C. The information is in the format `<address>.<mask>` `<data>`, to support the read-modify-write sequence. All commands are indexed, to facilitate multiple commands in a given block. Commands are sent to the device in sequence, starting from index 0, in the following format:

```
commands command-index .<addr>.<mask> = <data>;
```

P M peci ic ommands

The PWM specific commands are as follows:

ommand	Value
type	2 (for PWM 2)
controller-id	0 to 7 / Based on the platform /

source-frq-hz	PWM clock source freq / In Hz /
period-ns	Period in Nano Seconds / PWM periods /
min-microvolts	Vout from PWM regulator if duty cycle is 0
max-microvolts	Vout from PWM regulator if duty cycle is 100
init-microvolts	Vout from PWM regulator after initialization.
enable	1/0 / Enable the PWM or ust configure /

For example:

```
# 1. Set 950mV voltage.
pmic.core.3.block 0 .t pe = 2; # PWM T pe
pmic.core.3.block 0 .controller-id = 5; #GP_PWM6
pmic.core.3.block 0 .source-frq-h  = 102000000; #102M
pmic.core.3.block 0 .period-ns = 2600; # 384  is period.
pmic.core.3.block 0 .min-microvolts = 600000;
pmic.core.3.block 0 .max-microvolts = 1200000;
pmic.core.3.block 0 .init-microvolts = 950000;
pmic.core.3.block 0 .enable = 1;
```

Generic ormat

The following code snippets show the common and rail specific parameters in a generic format.

The common parameters are:

```
pmic.command-retries-count = <u32>;
pmic.wait-before-start-bus-clear-us = <u32>;
pmic.rail-count  = <u32>;
```

The rail-specific parameters are:

```
pmic.<rail-name>.block-count
```

The generic format is as follows:

```
##### BLOC  0 ####
pmic.<rail-name>.<rail-id>.block 0 .t pe = <0 for MMIO, 1 for I2C>
pmic.<rail-name>.<rail-id>.block 0 .dela  = <u32>
pmic.<rail-name>.<rail-id>.block 0 .count =  <calculated>;

#For I2C specific
pmic.<rail-name>.<rail-id>.block 0 .I2c-controller-id = <u32>;
pmic.<rail-name>.<rail-id>.block 0 .slave-add = <u32>;
pmic.<rail-name>.<rail-id>.block 0 .reg-data-si e = <u32>;
pmic.<rail-name>.<rail-id>.block 0 .reg-add-si e = <u32>;
```

```
#I2C and MMIOs

pmic.<rail-name>.<rail-id>.block 0 .commands 0 .<addr>.<mask> = <da
ta>;

pmic.<rail-name>.<rail-id>.block 0 .commands 1 .<addr>.<mask> = <da
ta>;

pmic.<rail-name>.<rail-id>.block 0 .commands 2 .<addr>.<mask> = <da
ta>;

pmic.<rail-name>.<rail-id>.block 0 .commands 3 .<addr>.<mask> = <da
ta0>;

::::


##### BLOC  1 ####

pmic.<rail-name>.<rail-id>.block 1 .t pe = <0 for MMIO, 1 for I2C>

pmic.<rail-name>.<rail-id>.block 1 .dela  = <u32>

pmic.<rail-name>.<rail-id>.block 1 .count = <Calculated>

#For I2C

pmic.<rail-name>.<rail-id>.block 1 .I2c-controller-id

pmic.<rail-name>.<rail-id>.block 1 .slave-add

pmic.<rail-name>.<rail-id>.block 1 .reg-data-si e

pmic.<rail-name>.<rail-id>.block 1 .reg-add-si e


#I2C and MMIOs

pmic.<rail-name>.<rail-id>.block 1 .commands 0 .<addr>.<mask> = <da
ta>;

pmic.<rail-name>.<rail-id>.block 1 .commands 1 .<addr>.<mask> = <da
ta>;

pmic.<rail-name>.<rail-id>.block 1 .commands 2 .<addr>.<mask> = <da
ta>;

pmic.<rail-name>.<rail-id>.block 1 .commands 3 .<addr>.<mask> = <da
ta0>;

::::
```

For example, in usage:

```
pmic.command-retries-count = 1;

pmic.wait-before-start-bus-clear-us = 0;

pmic.rail-count  = 6;


##############GENERIC RAIL  (ID = 1) DATA ###############

pmic.generic.1.block-count = 1;


# 1. Set PMIC MBLDP = 1, CNFGGLBL1 bit 6 = 1

pmic.generic.1.block 0 .t pe = 1; # I2C T pe

pmic.generic.1.block 0 .i2c-controller-id = 4;

pmic.generic.1.block 0 .slave-add = 0x78; # 7BIt:0x3c

pmic.generic.1.block 0 .reg-data-si e = 8;
pmic.generic.1.block 0 .reg-add-si e = 8;
pmic.generic.1.block 0 .dela  = 10;
pmic.generic.1.block 0 .count = 1;
pmic.generic.1.block 0 .commands 0 .0x00.0x40 = 0x40;

######################## #CORE RAIL  (ID = 3) DATA ###############
pmic.core.3.block-count = 2;

# 1. Set 950mV voltage.
pmic.core.3.block 0 .t pe = 1; # I2C T pe
pmic.core.3.block 0 .i2c-controller-id = 4;
pmic.core.3.block 0 .slave-add = 0x70; # 7BIt:0x38
pmic.core.3.block 0 .reg-data-si e = 8;
pmic.core.3.block 0 .reg-add-si e = 8;
pmic.core.3.block 0 .dela  = 1000;
pmic.core.3.block 0 .count = 1;
pmic.core.3.block 0 .commands 0 .0x07.0xFF = 0x2E;

# 2. Set GPIO3 Power down slot to 6.
pmic.core.3.block 1 .t pe = 1; # I2C T pe
pmic.core.3.block 1 .i2c-controller-id = 4;
pmic.core.3.block 1 .slave-add = 0x78; # 7BIt:0x3c
pmic.core.3.block 1 .reg-data-si e = 8;
pmic.core.3.block 1 .reg-add-si e = 8;
pmic.core.3.block 1 .dela  = 10;
```

PWM specific example:

```
pmic.core.3.block-count = 1;

# 1. Set 950mV voltage.
pmic.core.3.block 0 .t pe = 2; # PWM T pe
pmic.core.3.block 0 .controller-id = 5; #GP_PWM6
pmic.core.3.block 0 .source-frq-h  = 102000000; #102M
pmic.core.3.block 0 .period-ns = 2600; # 384  is period.
pmic.core.3.block 0 .min-microvolts = 600000;
pmic.core.3.block 0 .max-microvolts = 1200000;
pmic.core.3.block 0 .init-microvolts = 950000;
pmic.core.3.block 0 .enable = 1;
::::
```

Configuring Generic Rails

When configuring generic rails for PMIC, consider the following:

- Correcting the PMIC default configuration without the OTP is required. However, if the configuration is on the default OTP, reconfiguration is not required.
- To configure some devices in the system using I2C, platform specific configuration is not required.
- Write the configuration of devices, such as PMIC in P3310, then perform the configuration for the baseboard devices. This ensures that if the setup includes different configurations, all PMIC configurations for that P3310 is performed successfully.
- To assist in removing blocks if a module is not used, use a property comment in each block as follows:

  ```
  <module_name>:<device>
  ```

  For a P3310 module used for PMIC configuration:

  ```
  # P3310: PMIC: Set PMIC MBLDP = 1, CNFGGLBL1 bit 6 = 1
  ```

  For an expander configuration in the P25 7 baseboard:

  ```
  # baseboard (P2597): Expander: 5V0_ DMI_EN
  ```

Configuring Security Configuration Registers

Jetson TX1/TX2 devices have separate registers for configuring bridge client security, bridge firewalls, known as Security Configuration Registers (SCRs). SCRs are either configured by NVIDIA for Jetson TX1/TX2 platforms or re-configured for custom platforms. Custom configuration uses MB1 BCT at the MB1 stage.

The list of SCRs for re-configuration for custom platforms is the same in MB1 device-side code and SCR platform data. SCR register addresses, hard coded in MB1, can only contain data from the platform. The data cannot be masked, and can only be written to the registers as is.

The SCR configuration file is at:

```
<top>/bootloader/<platform|ver>/BCT/auto_scr.cfg
```

Usage

```
scr.<reg_index>.<exclusion-info> = <32 bit value>; # <reg_name>
```

Where:

- `scr` is the domain name prefix for the setting.
- `<reg_index>` is the matching MB1 and CFG file sequence, beginning at 0.
- `<exclusion-info>` is one of the following values:

Value	Description
0	**Include** regular SCRs loaded from BCT in cold boot, from stored context in warm boot.
1	Exclude: Present data in the CFG file but do not load data from the BCT. Allows SCR programming in MB2 or later.

2	SC7 resume: Program from BCT in cold boot, but exclude for warm boot.

MB1 code lists SCR register absolute addresses in an indexed list.

Example

```
# SCR register configurations
scr.134.1 = 0x20000000; # GPIO_M_SCR_00_0
scr.135.1 = 0x20000000; # GPIO_M_SCR_01_0
```

Miscellaneous Configurations

The miscellaneous configuration file is at:

```
<top>/bootloader/<platform|ver>/BCT/tegra186-mb1-bct-misc-quill-p23
82.cfg
```

Note	**Revie eac po er rail e ore customi ing t e carrier oard.**

The fields in `misc.cfg` are as follows:

ield	Description	on iguration xample
enable can boot	Controls early CAN initialization. If set, spe-can firmware loading spe-r5 processor boot is done.	enable can boot 1
enable blacklisting	Controls DRAM ECC blacklisting: 0: Disable ECC blacklisting 1: Enable ECC blacklisting	enable blacklisting 0
disable sc7	Controls SC7 state entry: 0: Enable sc7 1: Disable sc7	disable sc7 0
fuse visibility	Certain fuses cannot be read or written by default because they are not visible. If this field is set, MB1 enables fuse visibility for such fuses.	fuse visibility 1
enable vpr resize	Controls enablement of VPR resize functionality.	enable vpr resize 0
Disable el3 bl	Used for eliminate execution of EL3 Bl after secure os and can start EL2 bootloader	Disable el3 bl 1

A TAG

The AO-TAG register is programmed in MB1 which controls the maximum temperature Jetson TX1/TX2 systems are allowed to operate. If the temperature exceeds that limit, auto-shutdown is triggered.

A Tag ontrol ields	Description	on iguration xample
boot temp threshold	Boot temperature threshold in millicentigrade. If temperature is higher than the temperature specified in this field, MB1 waits or shuts down the device.	aotag.boot temp threshold 105000
cooldown temp threshold	Cool down temperature threshold in millicentigrade. MB1 resumes booting when the device has cooled to this threshold temperature.	aotag.cooldown temp threshold 85000
cooldown temp timeout	Contains max time MB1 should wait for system temperature to go down below cooldown temp threshold .	Cooldown temp timeout 30000
enable shutdown	If set to 1, enables shutdown using aotag if temperature is above boot temperature threshold.	aotag.enable shutdown 1

lock

The clock control fields hold the clock divider values for the various modules that MB1 programs.

lock ontrol ields	Description	on iguration xample
bpmp cpu nic divider	Program the cpu nic divider to control the BPMP CPU frequency. A value 1 less than the value in the field is directly written to the register.	clock.bpmp cpu nic divider 1
bpmp apb divider	Program the apb divider to control the APB bus frequency. A value 1 less than the value in the field is directly written to the register.	clock.bpmp apb divider 1
axi cbb divider	Program the axi cbb divider to control the AXI-CBB bus frequency. A value 1 less than the value in the field is directly written to the register.	clock.axi cbb divider 1
se divider	Program the se divider to control the SE Controller frequency. A value 1 less than the value in the field is directly written to the register.	clock.se divider 1

aon cpu nic divider	Program the cpu nic divider to control the AON(SPE) CPU frequency. A value 1 less than the value in the field is directly written to the register.	clock.aon cpu nic divider 1
aon apb divider	Program the apb divider to control the AON(SPE) APB frequency. A value 1 less than the value in the field is directly written to the register.	clock.aon apb divider 1
aon can0 divider	Program the can0 divider to control the CAN0 controller frequency. A value 1 less than the value in the field is directly written to the register.	clock.aon can0 divider 1
aon can1 divider	Program the can1 divider to control the CAN1 controller frequency. A value 1 less than the value in the field is directly written to the register.	clock.aon can1 divider 1
osc drive strength	Unused	-
pllaon divp	Program the P value of PLL-AON. A value 1 less than the value in the field is directly written to the register.	clock.pllaon divp 2
pllaon divn	Program the N value of PLL-AON. A value 1 less than the value in the field is directly written to the register.	clock.pllaon divn 25
pllaon divm	Program the M value of PLL-AON. A value 1 less than the value in the field is directly written to the register.	clock.pllaon divm 1

P Parameters

These settings contain the initial settings passed to CPU-Init FW. Do not change these settings.

ield	Description	on iguration xample
Bootcpu	Specify Boot CPU. 4 means A57 cpu0 and 0 mean Denver0. For automotive	cpu.bootcpu 4

	applications use A57-cpu0.	
ccplex platform features	Platform feature passed to the CPU-Init FW.	cpu.ccplex platform features 0x581
lsr dvcomp params b cluster	Contains setting for initializing ADC and DVC, which need to be functional before CPU rails are brought up	cpu.lsr dvcomp params b cluster 0xC0780F05C
lsr dvcomp params m cluster	Contains setting for initializing ADC and DVC, which need to be functional before CPU rails are brought up	cpu.lsr dvcomp params m cluster 0xC0780F05C
nafll m cluster data	Initial NAFLL settings for cluster for Denver	cpu.nafll m cluster data 0x11F04461
nafll b cluster data	Initial NAFLL settings for cluster for A57	cpu.nafll b cluster data 0x11F04461

A T ettings

The AST settings are for various firmware loaded by MB1/MB2. These are the virtual addresses of the firmware. MB1/MB2 programs corresponding physical addresses based on the location where it loaded the firmware in memory (DRAM). Normally there is no need to change these settings.

ields	Description	on iguration xample
bpmp fw va	Virtual address for BPMP-FW	ast.bpmp fw va 0x50000000
mb2 va	Virtual address for MB2-FW	ast.mb2 va 0x52000000
sce fw va	Virtual address for SCE-FW	ast.sce fw va 0x70000000
apr va	Virtual address for Audio-protected region used by APE-FW	ast.apr va 0xC0000000
ape fw va	Virtual address for APE-FW	ast.ape fw va 0x80000000

arveout

These settings specify the address and size for BL carveout.

ield	Description	on iguration xample
cpubl carveout addr	Start location of the CPU-BL Carveout	sw carveout.cpubl carveout addr 0x 6000000
cpubl carveout size		

	Size of the CPU-BL Carveout	sw carveout.cpubl carveout size 0x02000000
mb2 carveout size	Size of the MB2 Carveout	sw carveout.mb2 carveout size 0x00400000

De ug

The debug functionality can be enabled disabled using BCT flag.

De ug ontrol ields	Description	on iguration xample
uart instance	Configures the UART instance for console prints.	debug.uart instance 1
enable log	Enables/disables console logging.	debug.enable log 1
enable secure settings	Unused.	-

I2 ettings

These settings specify the operating frequency of the I2C bus in MB1/MB2 (default is 100 KHz).

ield	Description	on iguration xample
0	Specify the clock for I2C controller instance 0	i2c.0 400
4	Specify the clock for I2C controller instance 4	i2c.4 1000

Dev Parameters

These are the device settings used by MB1/MB2.

ield	Description	on iguration xample
qspi.clk src	Specify the clock source. The value corresponds to what is mentioned in the SPI CLK SRC register. 0: pllp out0 4: pllc4 muxed	devinfo.qspi.clk src 0
qspi.clk div	clk div N 1 Hence N 3 clk rate 163.2 MHz (408 MHz / ((N / 2) 1))	devinfo.qspi.clk div 4
qspi.width	Specify the width of the SPI BUS during transfer 0 : 1 bit (x1 mode) 1 : 2 bit (x2 mode) 2 : 4 bit (x4 mode)	devinfo.qspi.width 2
qspi.dma type	Specify which DMA to use for transfer if mode of	devinfo.qspi.dma type 1

	transfer is DMA. For SPI, in MB1/MB2, BPMP-DMA should be used. 0 : GPC-DMA 1 : BPMP-DMA	
qspi.xfer mode	Specify mode of transfer 0: PIO 1: DMA	devinfo.qspi.xfer mode 1
qspi.read dummy cycles	The dummy cycles allow the device internal circuits additional time for accessing the initial address location. During the dummy cycles the data value on IOs are don t care and may be high impedance.	devinfo.qspi.read dummy cycles
qspi.trimmer val1	tx clk tap delay for SPI	devinfo.qspi.trimmer val1 0
qspi.trimmer val2	rx clk tap delay for SPI	devinfo.qspi.trimmer val2 0

atc dog Timer ontroller ettings

These settings specify watchdog timer controller register values. These values will be configured by MB1.

ield	Description	on iguration xample
bpmp wdtcr	Contains the bpmp processer watchdog timer register value	wdt.bpmp wdtcr 0x710640 configures for 100sec
Sce wdtcr	Contains the SCE processer watchdog timer register value	wdt.sce wdtcr 0x707103
aon wdtcr	Contains aon s watchdog timer register value	wdt.aon wdtcr 0x700000
rtc2 ao wdtcr	Contains rtc2 ao watchdog timer register value	wdt.rtc2 ao wdtcr 0x700000
top wdt0 wdtcr	Contains top wdt0 watchdog timer register value	wdt.top wdt0 wdtcr 0x715016
top wdt1 wdtcr	Contains top wdt1 watchdog timer register value	wdt.top wdt1 wdtcr 0x710640
top wdt2 wdtcr	Contains top wdt2 watchdog timer register value	wdt.top wdt2 wdtcr 0x707103

PMIC

The PMIC configuration file is created manually as follows:

1. Get information about the set of commands to enable and setting voltage of each rail.
   - If OTP values are in desired voltage, do not reprogram voltage register.
   - Do not enable all rails, perform the recommended by boot sequence.

- The voltage for rail must be set per the boot sequence recommendation.

2. Get information about generic setting required from the MB1.
3. Once all information is collected, split these commands per rail.
4. Make the list of commands sequence, delay between commands and then make blocks. Blocks can contain multiple commands if they:
   - Are of the same type such as I2C or MMIO
   - Have the same delay
   - Communicate to the same device

   If anything is different, it will be in different blocks.
5. Create configuration file based on the above details.

BootROM

The BOOTROM configuration file is created manually as follows:

1. Get information about the set of commands to send to device in different reset path.
   - It is possible that the same type for commands are used for different reset paths. Collect all such information from system team.
2. Make the sets of commands required for each reset path independently.
3. Pickup commands for one reset path.
4. Make the list of commands sequence.
5. Delay between commands and then make blocks out of these. Blocks can contain multiple commands if they:
   - Are of the same type such as I2C or MMIO
   - Have the same delay
   - Communicate to same device

   If anyything is different, it will be in different blocks.
6. Put all blocks in one `aoblocks`.
7. Similarly, make all `aoblocks` for all reset paths.
8. Initialize the different reset paths `aocommand` with these `aoblock` indexes.

. Create configuration file based on above details.

Flashing

For information on flashing, refer to Flashing the Boot Loader and Kernel.