AN13400

i.MX 8M Low Power Design By M Core Running In System Suspend

Rev. 1 — 22 September 2022 **Application note**

Document information

Information	Content
Keywords	i.MX 8M, M core, low-power applications
Abstract	This application note aims to help to deploy low-power applications with the M core alive on the NXP i.MX 8M series SoCs.

1 Introduction

The i.MX 8M family of applications processors, based on Arm Cortex-A53, and Cortex-M cores, provide industry-leading audio, voice, and video processing for applications that scale from consumer home audio to industrial building automation and mobile computers.

As more users intend to use the Cortex-M core in their products for low-power purposes, this application note aims to help you to deploy low-power applications with the M core alive on the NXP i.MX 8M series SoCs.

It demonstrates how to minimize power consumption with changes on the kernel, the M core application, ATF, and U-Boot when the main system is set into Suspend mode.

The target audiences of the document are those users who want:

- Low-power requirement on i.MX 8M.
- To run both the Linux kernel and the M core application.
- To run the M core application only in TCM without DDR when the kernel is suspended.
- SoC IP peripherals' ownership shared/switched between cores.
- To become familiar with the expected processor power consumption in various scenarios.

The data presented in this application note is based on empirical measurements taken on a small sample; the presented results are not guaranteed.

2 Definitions, acronyms, and abbreviations

Table 1. Acronyms and meanings

Acronyms	Meanings
AMP	Asymmetric Multiprocessing
ATF	Arm Trusted Firmware
DTB	Device Tree Blob
DTS	Device Tree Source
DSM	Deep Sleep Mode
DVFS	Dynamic voltage and frequency scaling
GIP & GIPn	General Interrupt Request n Pending
GIR & GIRn	General Purpose Interrupt Request n
GPC	General Power Management
HMP	Heterogeneous Multicore Processing
LPA	Low-Power Audio
MU	Message Unit
RPMsg	Remote Processor Messaging
RTOS	Real-Time Operating System
SoC	System on Chip
TCM	Tightly Coupled Memory

3 Overview of i.MX 8M Low Power

This section gives general information about i.MX 8M Low Power.

3.1 Voltage supplies

The i.MX 8M series processors have several power supply domains (voltage supply rails) and internal power domains. The figure below shows the connectivity of these supply rails and the distribution of the internal power domains on i.MX 8MP.

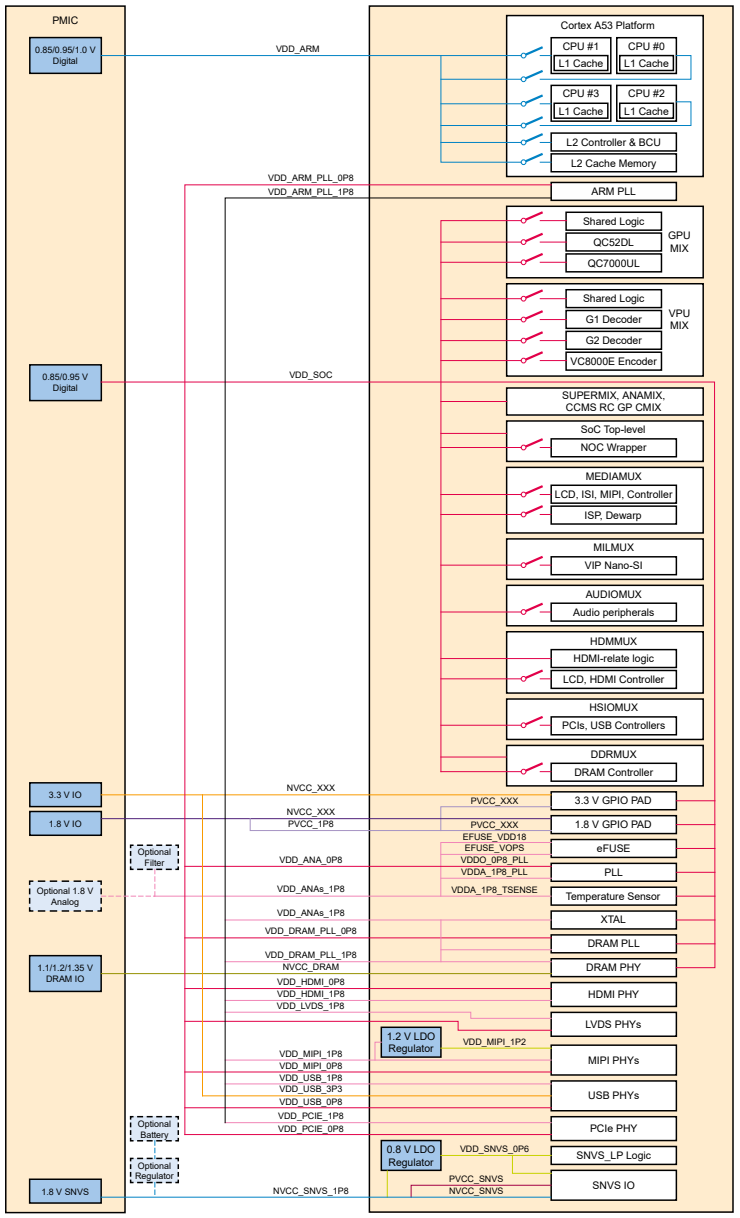

Figure 1. i.MX 8MP power rails

Note: *For the recommended operating conditions of each supply rail and for a detailed description of the groups of pins that are powered by each I/O voltage supply, see*

the SoC data sheet. For more information about the power rails, see Chapter "Power Management Unit (PMU)" in the Reference Manual.

3.2 Low-power modes on i.MX 8M

This section describes the specifics of low-power modes on i.MX 8M and the difference between Suspend mode and DSM (Deep Sleep mode).

3.2.1 SoC low-power modes introduction

There are two platform low-power modes, **Wait** and **Stop,** and one system low-power mode, **DSM**.

Wait and Stop modes are supported by both CPU platforms (each representing a CPU domain): the Quad-core Cortex A53 platform and the Cortex M7 platform.

In our standard release, only Stop mode is supported.

Note: *There are two types of Wait and Stop modes, Fast-Wake-up and Non-Fast-Wake-up. The default is Non-Fast-Wake-up. You can only use one of them. The difference is that during Fast-Wake-up, the clock, and PLL (Phase-locked loop) management are handled by software.*

Non-Fast-Wake-up, Wait, and Stop are configured in `GPC_LPCR_A53_BSC`, `GPC_LPCR_A53_BSC2`, *and* `GPC_LPCR_M7` *registers. Fast-Wake-up, Wait, and Stop are configured in the* `GPC_SLPCR` *register.*

DSM is a system low-power mode. When entering DSM, hardware can help to switch the voltage of VDD_SOC from Run to DSM and shutdown all PLLs and clocks.

For other low-power modes, you must optimize the clocks and PLLs using software.

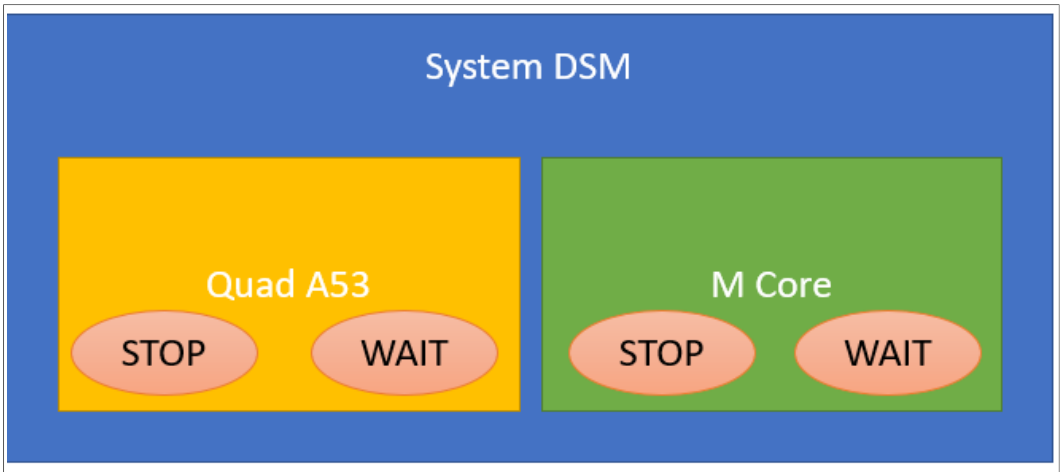

Figure 2. Low-power modes' relationship

The system goes into DSM under the following conditions:

- Both A53 and M7 are in Stop mode (Non-Fast-Wake-up Stop).
- Both `GPC_SLPCR[EN_A53_FASTWUP_STOP_MODE]` and `GPC_SLPCR[EN_M7_FASTWUP_STOP_MODE]` are not set.
- `GPC_SLPCR[EN_DSM]` is set.

- All PLLs are closed from CCM configuration, otherwise the system is not able to wake up.

Note: *The system can enter DSM either from the A core side or the M core side. See the note in RM:*

If `GPC_LPCR_M7[MASK_DSM_TRIGGER]` *is set, the system goes into DSM when A53 goes into Stop mode and* `GPC_SLPCR[EN_A53_FASTWUP_STOP_MODE]` *is not set. If* `GPC_LPCR_A53_BSC[MASK_DSM_TRIGGER]` *is set, the system goes into DSM when M7 goes into Stop mode and* `GPC_SLPCR[EN_M7_FASTWUP_STOP_MODE]` *is not set.* `GPC_LPCR_M7[MASK_DSM_TRIGGER]` *and* `GPC_LPCR_A53_BSC[MASK_DSM_TRIGGER]` *cannot be set at the same time.*

For more details, refer to the GPC chapter in SoC RM.

3.2.2 Suspend and DSM (Deep Sleep mode)

This section describes the difference between Suspend mode and DSM.

General introduction:

- DSM is a system low-power mode supported by SoC.
- In Linux, you can put the kernel into Suspend mode using the command "`echo mem > /sys/power/state`". On i.MX 8M, it puts core A53 into Stop mode. When A53 is in stop and the M core is not running, the system will enter DSM directly. It is the standard behavior supported in the standard release.

4 Application design

This section describes the specifics of the application used.

4.1 Why use the M core for low-power cases

You can use the M core in your applications for:

- Lower power consumption comparing to the A core.
- Faster wake-up from IRQ. It is very useful in monitoring tasks.
- Faster interrupt handler.

4.2 Application scenario

This application scenario is:

1. The Linux Kernel runs on Cortex®-A53, and M core applications (no matter with BareMetal or RTOS) can work separately.
2. In low-power mode, the kernel enters Suspend-to-Ram mode. It is in the clock-gating, low-voltage mode while the Cortex-M subsystem performs low-power, real-time system monitoring tasks.
3. The M core application can wake up the Linux OS running on the Cortex-A53 core.
4. The M core application only runs in TCM with DDR in Retention mode when the kernel is suspended.

4.3 Make the M core alive when the A core is in Suspend mode

There are two ways to keep the M core alive when the A core is suspended:

- Change the clock source of UART used by the M core to 24 MHz OSC.
- Define LPA Flags (see Enable low-power audio flags) in the M core application that puts the A core into Fast-Wake-up-Stop mode (`GPC_SLPCR[EN_A53_FASTWUP_STOP_MODE]` is set).

4.3.1 Method 1: Change clock source of UART used by the M core to 24 MHz OSC

It is the simplest way to keep the M core alive when the A core is suspended.

In this scenario:

- All PLLs are bypassed and therefore all PLLs used by the M core are changed to 24 MHz.
- The M core source clock is changed to 24 MHz.
- DRAM is in Retention mode.

As all PLLs are OFF, optimization is not needed.

4.3.2 Method 2: Define LPA flags in the M core application

When LPA flags are defined, the ATF code puts the A core into Fast-Wake-up-Stop mode.

In this scenario:

- The frequency of all PLLs is kept.
- The clocks of all modules are kept.
- The frequency of the M core is kept.
- DRAM can be defined as retention or not.

So, when LPA flags are defined to put the A core into Fast-Wake-up-Stop mode, you have more performance on the M core. But as all PLLs and clocks are kept, the power is very high.

Some optimizations are needed here.

4.3.3 Define the scenario to know how to use the M core

In practical applications, you must define their low-power scenario first.

Method 1 is easier, but it has more limitations. Method 1 uses 24 MHz as the core and peripherals' clock source. In case your applications do not need high performance and 24 MHz is enough for the core and peripherals, use method 1.

Method 2 is more flexible, but requires software optimization to shut down unused PLLs and clocks.

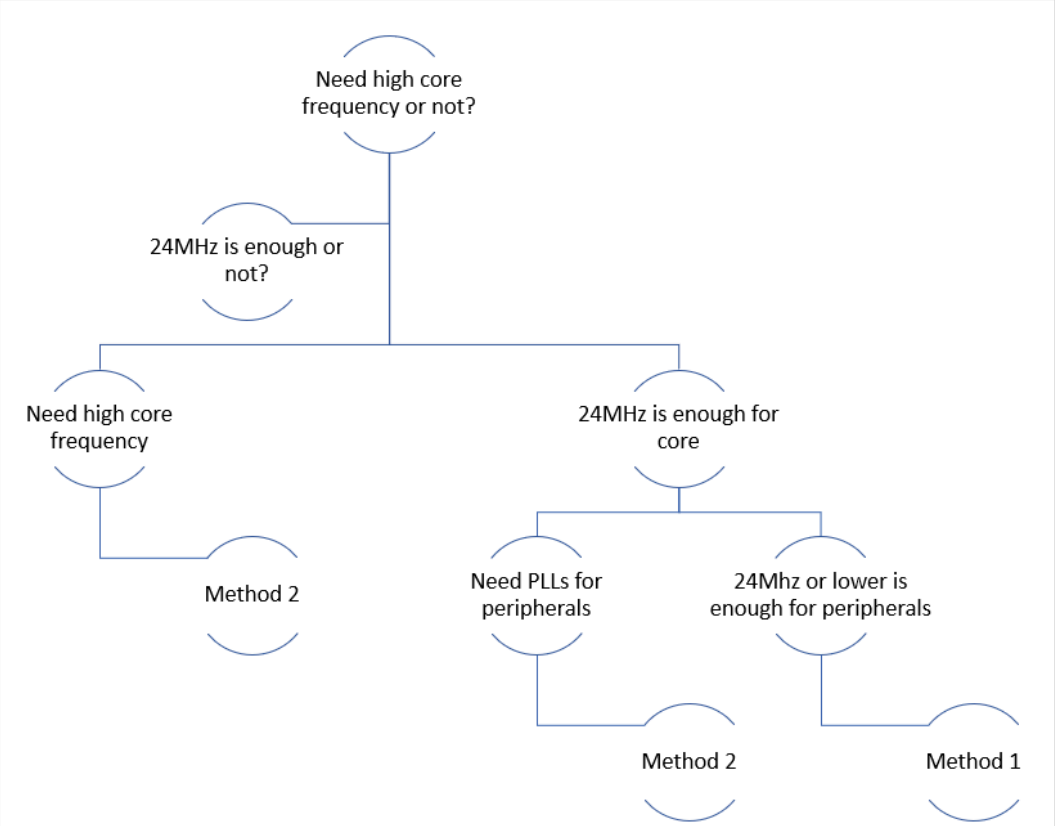

Figure 3. Select the method for an M core low-power application

4.4 Diagram of use case scenarios

This section shows the difference in use case scenarios when the A core is in Run and in Suspend modes.

4.4.1 The A core in Run mode

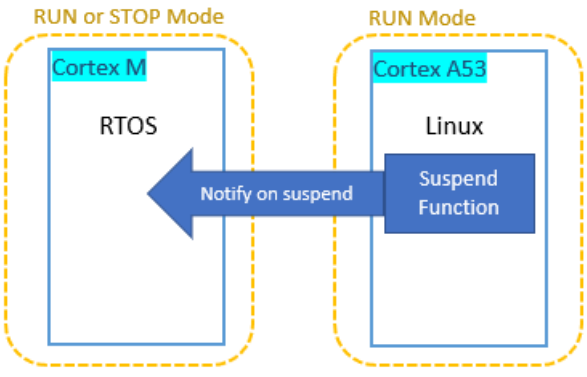

Figure 4. Cortex-A53 Run and Enter suspend

When the A core is running, the M core can be in Run or Stop mode.

In this state, the kernel can enter Suspend mode using the following command:

```
"$: echo mem > /sys/power/state"
```

In this application, when the kernel enters Suspend mode, ATF sends a notification to the M core.

4.4.2 The A core in Suspend mode

When the M core image receives a notification of the kernel Suspend mode, it enters low-power mode.

As shown above, the application might do some low power monitoring tasks.

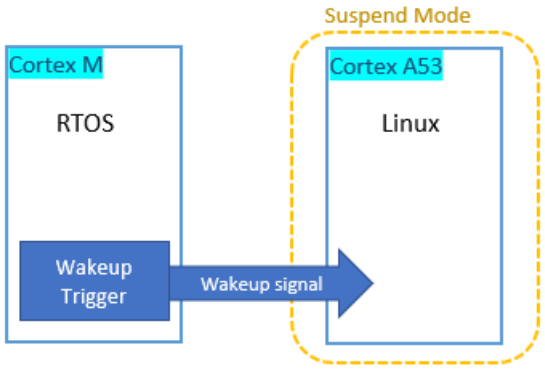

Figure 5. Cortex-M wakes up Cortex-A53

When the wake-up condition matches, the Cortex-M core triggers GIRn to send a wake-up interrupt to the Cortex-A core. The A core exits from Suspend mode then.

4.5 Peripheral control in Cortex-M

On i.MX 8M, peripherals are shared between the Cortex-A core and the Cortex-M core, such as GPIO, I^2C, and UART.

In HMP applications, there are three cases of inter-core peripheral control.

4.5.1 Peripherals are owned exclusively by one core

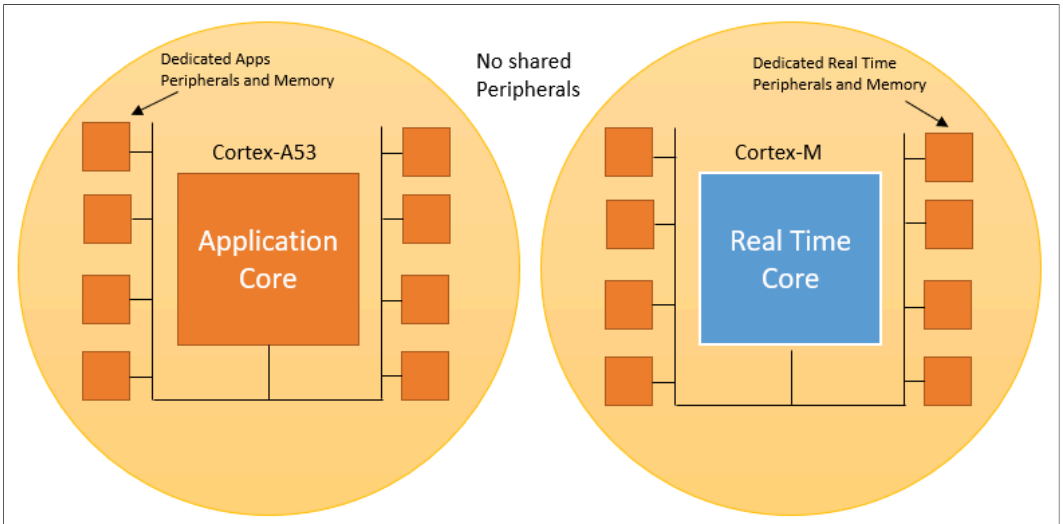

Figure 6. Exclusive modules in each core side

In this case, the Cortex-A core releases its control over these IPs, so that the Cortex-M core can fully manage them.

4.5.2 Some peripherals are shared

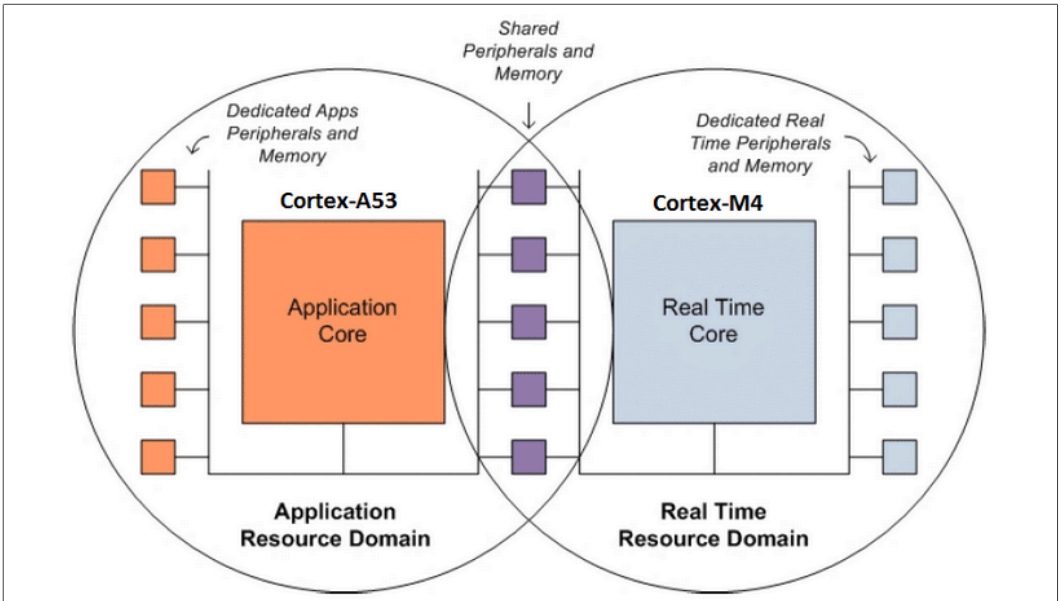

Figure 7. HMP application with shared peripherals

For those shared peripherals, there are two ways when both the A core and the M core need to access them.

- The use of RPMsg.
- The use of practical drivers and time-division access.

4.5.2.1 The use of RPMsg

In the standard GA release and SDK package, RPMsg is used for inter-core communication.

The M core uses a practical driver to control the peripheral and the A core uses an RPMsg-based virtual driver, for example, the RPMsg I^2C driver, which sends control messages to the M core to control the hardware.

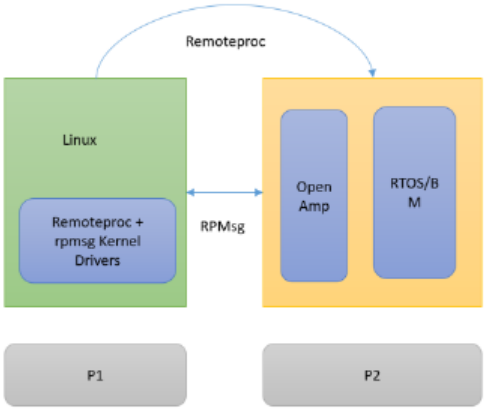

Figure 8. Use RPMsg to control the M core peripherals

In this way, the Cortex-A core also releases control over these IPs in its dts file and lets the Cortex-M core fully manage them.

See RPMsg Messaging Protocol for details.

It is the most recommended way but very slow in implementing a virtual driver.

If you do not have the experience or time to write the virtual RPMsg driver in the kernel, you can use another way. It is the use of drivers in the time-division way on both cores.

4.5.2.2 The use of practical drivers and time-division access on both cores

In this case, the time-division access means:

- There are practical control drivers in both the A core and the M core sides. Each driver controls the peripheral in different period, that is, Cortex-A controls the peripheral in Run mode and Cortex-M does it when the A core is suspended.
- It is an exclusive access for both drivers to the peripheral. The control of the peripheral switches between the Cortex-A and the Cortex-M sides.

Suggestions for implementing such cases are:

- Reinitialize modules when module control switches.
- Reinitialize module clocks' frequencies. It is optional if modules' clocks are kept ON in ATF changes.
- Do not disable the module's CCGR in CCM. Disabling the module's CCGR on the M core side can cause the kernel to crash or the module not to work after the A core wake-up.
- Use a hardware semaphore to avoid concurrent access.

Another issue is that the SDK packages only provide limited drivers, for example, i2c, spi, gpio, and so on. For other drivers, you may need to port it from other SoCs' SDK drivers,

for example, i.MXRT, i.MX8, and so on. For example, enet drivers in the example code are ported from the i.MXRT SDK package.

4.5.2.3 Comparison of RPMsg and time-division drivers

The advantages and disadvantages of both approaches are obvious.

Table 2. Comparison of RPMsg and time-division drivers

	RPMsg	Time-division drivers
Advantages	• Fewer potential concurrent access issues. • Simple mechanism. One practical driver, one virtual driver.	• DDR is not needed. The whole program can be in TCM. • Direct control. Each driver can control the module directly. • No need for a kernel virtual driver.
Disadvantages	• A bit slower as control messages are sent via RPMsg. • Need DDR. DDR cannot be in retention in Suspend mode. • Need an additional virtual RPMsg driver in the kernel.	• More effort in debugging concurrent access issues.

Note:

1. *Applications that do not need to use DDR when the A core enters Suspend mode can still use RPMsg and DDR when both cores are running.*
2. *The system can get stuck if the M core application tries to access DDR when it is in retention.*

4.5.2.4 Debugging tips for time-division drivers

Debugging time-division drivers on both cores requires more effort to handle two cases:

1. Concurrent access. When both drivers in each core side attempt to access the same module at the same time, the module driver might crash. For example, if both the Linux I^2C driver and the M core I^2C driver attempt to access I^2C0 at the same time, the I^2C driver (either in the kernel or in the M core) might crash.
2. Control switch after wake-up. When the kernel is suspended and awakened, control of the module is switched to the other core. The module may not work after switching.

For such cases, there are several suggestions and tips:

1. Check the suspend and resume function in the kernel module driver. You can re-initialize modules when module control switches, the suspend function must disable the module and clocks and the resume function must reset the module. Check if suspend/resume logic is correct.
2. While debugging, disabling the module's CCGR on the M core side can cause the kernel to crash or the module not to work after the A core wake-up. So, if the module does not work properly after wake-up, try to keep CCGR on when switching control switches.
3. Check clocks. Make sure that module clocks are correct before and after wake-up. In the kernel, check the clock summary using the "`cat /sys/kernel/debug/clk/clk_summary`" command.

5 Power optimization

The power optimization is for cases where a higher core frequency is required, and the M core must keep any PLLs for peripherals.

To optimize the power consumption of these use cases, use the items below:

Table 3. Power optimization

Item	General description
Kernel (optional)	Optional change for i.MX8 MQ and if the kernel RPMsg doorbell mechanism is used.
U-Boot	Reduce VDD_SOC RUN voltage.
ATF (Arm Trusted Firmware)	Disable or bypass PLLs, root clocks, CCGRs on VDD_SOC.
M core application	Choose proper clocks and LPA flags for the convenience of optimization.

5.1 Kernel optimization

The kernel optimization is optional as the Linux code in GA release can reach the lowest power if only the Linux kernel is needed.

Note:

For i.MX 8MQ, when the kernel enters Suspend mode and the M core is running, VDD_SOC might be very high as the clock of DCSS is still ON. In this case, disable the clock when the A core enters Suspend mode.

For the code change and other details, see How to Reduce SoC Power when Running M4 with A53 on i.MX8M (document: AN12225).

5.2 The M core application optimization

The M core application must define its functionality either when the kernel is running or the kernel is suspended. The paper focuses on the items that must be considered in the M core application when the kernel is suspended.

Regarding low power, there are three things to consider:

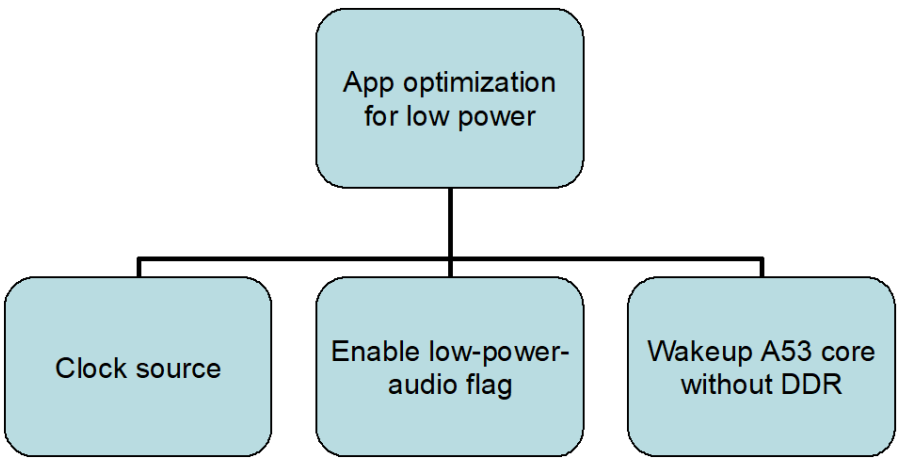

Figure 9. The M core application optimization items

5.2.1 Clock source

Since it is a low-power application, there are several rules for the M core application clock selection.

1. Consider using a 24 MHz crystal as a clock source.
   - If the system load is not high, use a 24 MHz crystal as the M core clock.
   - Use a 24 MHz crystal as the UART clock source.
2. Use PLLs as little as possible. You can disable PLLs in this way:
   In default SDK release's demos, several PLLs are used, for example, SYSPLL1 for the M core, SYSPLL2 for UART, audio PLLs are enabled. In a low-power application, to use fewer PLLs, use SYSPLL1 for the M core, UART, and disable other PLLs. In the low-power demo of this paper, a 24 MHz crystal is used as the clock source for the M core and UART. In this way, all PLLs can be disabled in the ATF code.

5.2.2 Enable low-power audio flags

Low-power audio flags are defined in the ATF code for the low-power audio demo in the SDK release.

When ATF detects these flags, all PLLs are not disabled, and the system enters Fast-Wake-up-Stop mode. Setting these flags does not impact any audio-related (sai port, sai clock, and so on) settings, but does affect PLLs, low-power mode, and so on.

In this way, the M core application can be in Run mode when the kernel enters Suspend mode.

In the low-power application, these flags are used to keep the M core alive when the A core is in DSM.

Examples of code and comparison of differences:

1. The M core application changes and the ATF code. On iMX8MP, the LPA flag register is `SRC-GPR10` but on other iMX8M SoCs, it is `SRC->GPR9`.

Table 4. The M core application changes and the ATF code

The M core application	ATF

`#define ServiceFlagAddr SRC->GPR9`
`#define ServiceBusy (0xDU)`
`#define ServiceIdle (0x0U)`
…
`void app_task(void *param)`
`{`
`ServiceFlagAddr = Service Busy;`
…
	`#define M4_LPA_ACTIVE 0x5555`
	`#define DSP LPA ACTIVE 0xD`
	`#define DSP_LPA_DRAM_ACTIVE 0x1D`
	`#define M4 LPA IDLE 0x0`
	…
	`bool imx_m4_lpa_active(void)`
	`{`
	`uint32_t lpa_status;`
	`lpa status = mmio read 32(IMX SRC BASE +`
	`LPA_STATUS);`
	`return (lpa_status == M4 LPA ACTIVE ||`
	`lpa status == DSP LPA ACTIVE || lpa status`
	`==`
	`DSP LPA DRAM ACTIVE);`
	`} …`

2. Low-power-audio flag descriptions. Currently, ATF supports 3 low-power audio flags, `M4_LPA_ACTIVE`, `DSP_LPA_ACTIVE`, , and `DSP_LPA_DRAM_ACTIVE` . The descriptions in the GA release are listed below:

Table 5. The descriptions in the GA release

	RAM status	Low-power mode	PLLs status	MU wake-up status
M4_LPA_ACTIVE DSP_ LPA_DRAM_ACTIVE	ON	Fast-Wake-up-Stop	ON	ON
DSP_LPA_ACTIVE	**Retention**	Fast-Wake-up-Stop	ON	ON

All three flags can keep the M core running, while the A core enters Suspend mode. One difference is that `DSP_LPA_ACTIVE` puts DRAM into Retention mode. For low-power purposes, we set the LPA flag to `DSP_LPA_ACTIVE`.

5.2.3 Wake-up

The M core application can wake up the A core when needed.

Table 6. The M core application changes and the ATF code

The M core application code to wake up the A core	The ATF code to enable MU wake-up
MU_TriggerInterrupts(MUB, kMU_ GenInt0InterruptTrigger);	`/* enable the MU wakeup */ if (imx_is_m4_enabled()) mmio clrbits 32(IMX GPC BASE + gpc_imr_offset[last_core] + 0x8, BIT(24));`

5.3 GIR mechanism

GIR stands for General Purpose Interrupt Request feature in MU. It is a bit field in the MU module.

TR (Transmit Register) and RR (Receive Register) registers can also be used for wake-up and notification. But we choose GIR as a wake-up and notification source for three reasons:

- Align with the kernel. The kernel uses GIR bits as Doorbell in mailbox driver. Using GIR has a better expansibility.
- Data requirement. In this demo, there is no data transfer. One-bit signal is enough.
- TR and RR are used in the kernel. TR registers are used to transfer data in the kernel. Using TR might lead to unnecessary issues.

The main diagram is shown below.

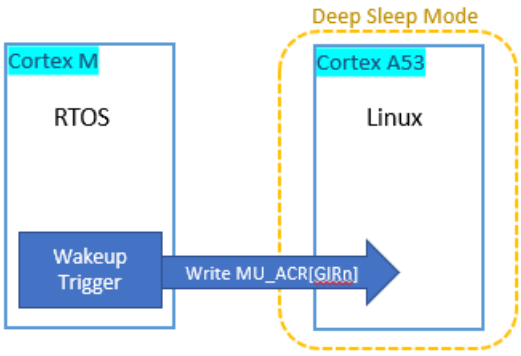

Figure 10. Cortex-M wakes up Cortex-A53

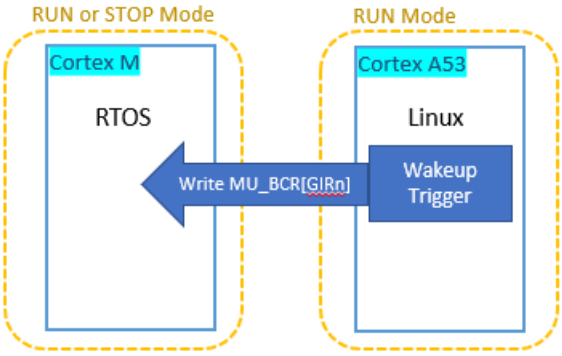

Figure 11. Cortex-A53 notifies that Cortex-M is in Suspend mode

5.3.1 GIR descriptions

In the registers of the MU module, three fields are related to GIR, SR[GIPn], CR[GIEn], and CR[GIRn].

1. CR[GIRn]

Table 7. CR[GIRn]

CR[GIRn]	Description
19-16 GIRn	For n = {0, 1, 2, 3} Processor B General Purpose Interrupt Request n. (Read-Write) • Writing "1" to the GIRn bit sets the GIPn bit in the ASR register on the Processor A-side. If the GIEn bit in the ACR register is set to "1" on the Processor a-side, a General Purpose Interrupt n request is triggered. • The GIRn bit is cleared if the GIPn bit (in the ASR register on the Processor A-side) is cleared by writing it (GIPn bit) as "1", thereby signaling the Processor B that the interrupt was accepted (cleared by the software). The GIPn bit cannot be written as "0" on the Processor B-side. • To ensure proper operations, you must verify that the GIRn bit is cleared (meaning that there is no pending interrupt) before setting it (GIRn bit). • GIRn bit is cleared when the MU resets. 0 Processor B General Interrupt n is not requested to the Processor A (default). 1 Processor B General Interrupt n is requested to the Processor A.

With these GIR bits, you can send a one-bit signal to MU from the other side. When a GIR bit is set, a GIP bit is set accordingly.

2. CR[GIEn]

Table 8. CR[GIEn]

CR[GIEn]	Description
31-28 GIEn	For n = {0, 1, 2, 3} Processor B General Purpose Interrupt Enable n. (Read-Write) • GIEn bit enables Processor B General Interrupt n. • If GIEn bit is set to "1" (enabled), then a General Interrupt n request is issued when the GIPn bit in the BSR register is set to "1". • If GIEn is cleared (disabled), then the value of the GIPn bit is ignored and no General Interrupt n request is issued • GIEn bit is cleared when the MU resets. 0 Disables Processor B General Interrupt n. (default) 1 Enables Processor B General Interrupt n.

It is related to an interrupt on the other side. Also, you must set it for wake-up purposes. To wake up, besides enabling GIE, enable MU in the GPC module. Normally, the MU wake-up is enabled in ATF when LPA is detected. The code is in `plat/imx/imx8m/<soc>/gpc.c`:

```
void imx set sys wakeup(unsigned int last core, bool pdn)
{
 … …
     /* enable the MU wakeup */
     if (imx_m4_lpa_active())

 mmio_clrbits_32(gpc_imr_offset[last_core] + 0x8, BIT(24));
}
```

But if an LPA flag is not set in the M core application, enable MU wake-up in the GPC module in the M core application code.

3. SR[GIPn]

Table 9. 3. SR[GIPn]

SR[GIPn]	Description
31-28 GIPn	For n = {0, 1, 2, 3} Processor B General Interrupt Request n Pending (Read-Write) • GIPn bit signals the Processor B that the GIRn bit in the ACR register on the Processor A-side was set from "0" to "1". If the GIEn bit in the BCR register is set to "1", a General Interrupt n request is issued. • The GIPn bit is cleared by writing it back as "1". Writing ""0" or writing ""1" when the GIPn bit is cleared is ignored. Use this feature in the interrupt routine, where the GIPn bit is cleared to de-assert the interrupt request source at the interrupt controller. • GIPn bit is cleared when the MU is reset. 0 Processor B general-purpose interrupt n is not pending. (default) 1 Processor B general-purpose interrupt n is pending.

SR[GIPn] is set when the remote writes CR[GIRn]. This bit must be read in time to de-assert interrupt.

5.3.2 GIR changes

There are two approaches to using GIR as a notification and wake-up source between cores.

Table 10. Approaches to using GIR as a notification and wake-up source between cores

	Write and clear GIR bit in ATF (Recommended)	Use mailbox driver in the kernel (Optional)
Code location	ATF	Linux Kernel
Advantages	• Direct. Access the GIR register directly • Easier. Do not need to follow kernel API rules. • Independent of versions. Can be added to every ATF version.	• Flexible. Use DTS to control the GIR bit index is easy. • Can be integrated with the module driver suspend function. For example, enet suspend function. • Faster. As it can be integrated into the module driver, the M core application can receive a suspend signal when the kernel is executing module suspend functions.
Disadvantages	• Not flexible. If the GIR bit index is changed, must modify ATF. • Must implement the GIR functions. • Slower. The M core application receives a suspend signal before WFI.	• Must study mailbox subsystem and follow the mailbox APIs. • Depends on the kernel version. The i.MX kernel supports mailbox driver since the 5.x kernel. For the 4.x kernel, an additional patch for GIR is needed.

Table 10. Approaches to using GIR as a notification and wake-up source between cores
...continued

	Write and clear GIR bit in ATF (Recommended)	Use mailbox driver in the kernel (Optional)

Code changes	See Use GIR for wake-up and notification for detailed changes.	DTS change (take FEC as example):
		`+/* request mailbox: rxdb, channel 0, txdb, channel 0, rxdb for receive interrupts, txdb for send notifications */`
		`+&fec {`
		`+ fsl,switch-mcore-ctrl;`
		`+ mbox-names = "rxdb", "txdb";`
		`+ mboxes = <&mu 3 0 &mu 2 0>;`
		`+ status = "okay";`
		`+};`
		Driver changes, see patch `0001-Switch-enet-control-between-A-core-and-M-core-for-cu.patch` in the attachment.

This example uses the GIR record bit in the ATF code.

5.4 U-Boot optimization

The voltage of VDD_SOC and VDD_ARM is configured in U-Boot.

From the data sheet, VDD_SOC can be nominal or overdrive voltage, VDD_ARM can be nominal, overdrive, or super-overdrive.

In the release, VDD_ARM is adjusted by DVFS in the kernel. So, you can only change the voltage of VDD_SOC.

In `board/freescale/<board> /spl.c` , both Run, and DSM voltage of VDD_SOC are configured.

The Fast-Wake-up-Stop mode (not DSM) is used in this case. In this mode, the Run voltage is applied.

Thus, to reduce the power, you must adjust the Run voltage of VDD_SOC from 0.95 V (Overdrive Voltage) to 0.85 V (Nominal Voltage) in U-Boot.

Note: *The Nominal and Overdrive voltage may vary for i.MX 8M family SoCs. Refer to data sheet for details.*

Example for i.MX 8MP which uses PCA9450:

```
diff --git a/board/freescale/imx8mp_evk/spl.c b/board/
freescale/imx8mp_evk/spl.c
index b26f5321bb..2c0081081d 100644
--- a/board/freescale/imx8mp_evk/spl.c
+++ b/board/freescale/imx8mp_evk/spl.c
@@ -193,7 +193,7 @@ int power_init_board(void)
         * Enable DVS control through PMIC_STBY_REQ and
         * set B1_ENMODE=1 (ON by PMIC_ON_REQ=H)
```

```
          */
-       pmic_reg_write(p, PCA9450_BUCK1OUT_DVS0, 0x1C);
+       pmic_reg_write(p, PCA9450_BUCK1OUT_DVS0, 0x14);
        pmic_reg_write(p, PCA9450_BUCK1OUT_DVS1, 0x14);
        pmic_reg_write(p, PCA9450_BUCK1CTRL, 0x59);
```

As for bd71837, the voltage levels are the same.

In power measurement, it can save about 20 mW power of VDD_SOC on i.MX 8MP.

5.5 ATF optimization

On i.MX 8M, ATF puts the system into low-power mode. So, ATF is the most important part of the optimization.

Perform PLL-related optimization at this step.

5.5.1 Overview

The digital logic inside the chip is supplied with two supplies: VDD_ARM and VDD_SOC.

- VDD_ARM is for the Cortex®-A53 platform.
- VDD_SOC is for the rest of the modules in SoC.

In board design, VDD_SOC provides power for all peripheral modules.

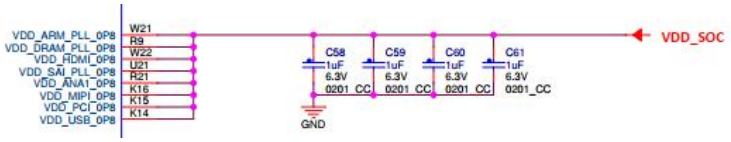

Figure 12. VDD_SOC

As the aim is to reduce modules' power consumption, the power optimizations are mainly for VDD_SOC.

All optimization points are listed below. The GIR change below is for the applications.

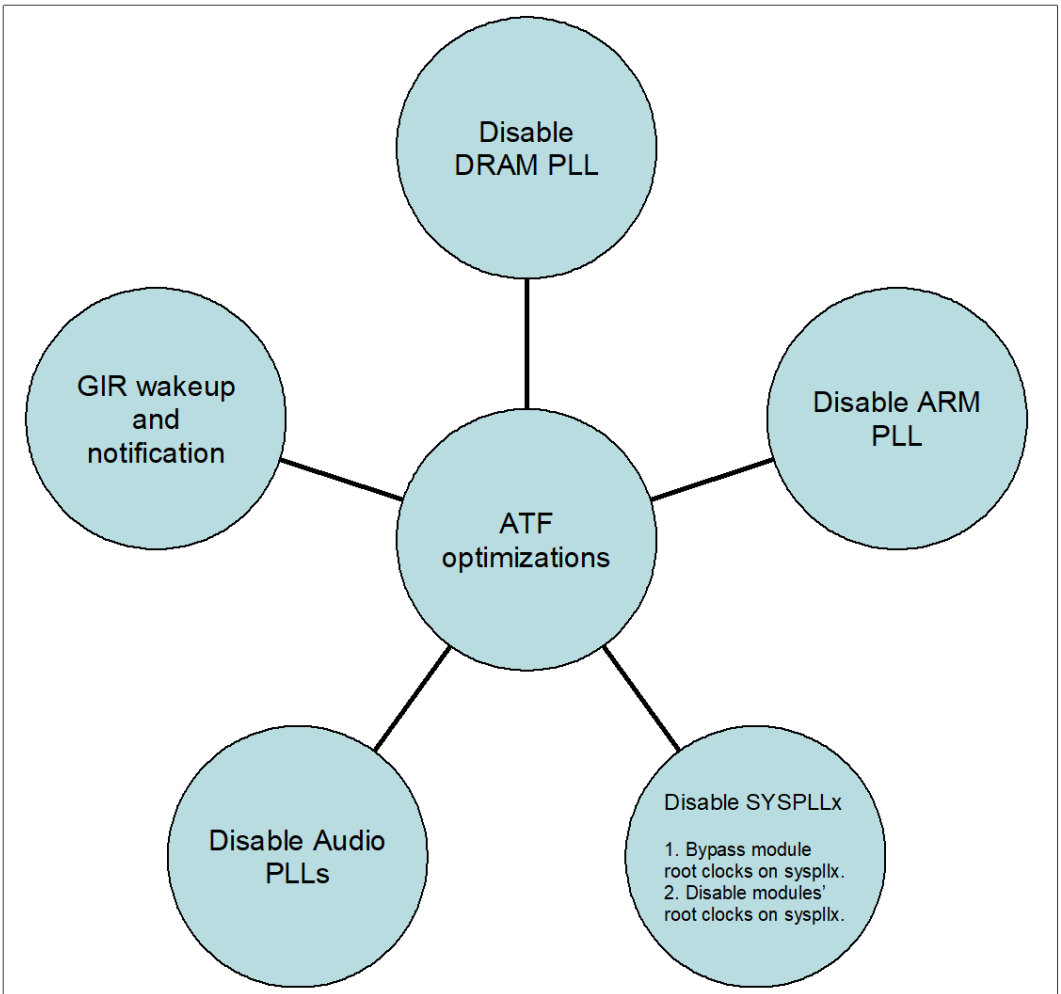

Figure 13. ATF optimization items

5.5.2 Use GIR for wake-up and notification

The drawback of RPMsg in low power is that the RPMsg needs DDR to be active to store the virtual queue structure. To keep it functioning at a low frequency to save power, the solution described in this document directly uses a GIR signal in MU triggered from the M core side, which bypasses the DDR.

In this case, GIR bits in MU can be used for below purposes:

1. Wake-up. When the A core enters the Fast-Wake-up-Stop mode, the M core can write a GIR bit to wake up the A core.
2. Notifications. The A core or the M core can write a GIR bit to send a notification to a remote core. The notification definition is defined by the user.

The code to wake up the A core and notify the M core in Suspend mode are added in ATF.

Example:

```
+static void imx_notify_m4_set_db(void)
```

```
+{
+       /* Use GIR[0] and enable interrupt */
+       mmio_setbits_32(IMX_MU_BASE + 0x24, BIT(19) | BIT(31));
+}
+
+static void imx_notify_m4_clear_db(void)
+{
+       /* Clear GIP[0] */
+       mmio_setbits_32(IMX_MU_BASE + 0x20, BIT(31));
+}
+
…
+                       NOTICE("notify m4 by setting db! \n");
+                       imx_notify_m4_set_db();
…
+                       NOTICE("clear db! \n");
+                       imx_notify_m4_clear_db();
…
```

5.5.3 Disable DRAM PLL

DDR consumes most power on VDD_SOC. The ATF code puts DDR into Retention before entering Suspend mode. But in the current implementation, DRAM PLL is kept on.

In testing on i.MX 8MP, disabling DRAM PLL can save about 400-600 mW power on VDD_SOC.

Check if DRAM PLL is disabled when DDR enters Retain mode in the ATF code.

Note: *In a standard release, as the system enters DSM, hardware helps to shut down PLLs (including DRAM PLL) and other clocks. But when the system enters Fast-Wake-up-Stop, it must be done by software.*

The patch below shows how to disable DRAM PLL:

```
diff --git a/plat/imx/imx8m/ddr/dram_retention.c b/plat/imx/
imx8m/ddr/dram_retention.c
index 685526f4b..260cacc7a 100644
--- a/plat/imx/imx8m/ddr/dram_retention.c
+++ b/plat/imx/imx8m/ddr/dram_retention.c
@@ -108,6 +108,10 @@ void dram_enter_retention(void)
        mmio_setbits_32(IMX_GPC_BASE + DDRMIX_PGC, 1);
        mmio_setbits_32(IMX_GPC_BASE + PU_PGC_DN_TRG,
 DDRMIX_PWR_REQ);
+       /* disable the DRAM PLL */
+       /* disabling DRAM PLL will save about 437mW, 667 => 230
 */
+       mmio_clrbits_32(IMX_ANAMIX_BASE + 0x50, BIT(9));
+
        VERBOSE("dram enter retention\n");
 }
@@ -134,6 +138,9 @@ void dram_exit_retention(void)
        mmio_setbits_32(IMX_GPC_BASE + PU_PGC_UP_TRG,
 DDRMIX_PWR_REQ);
        mmio_write_32(SRC_DDR1_RCR, 0x8F000006);
+       /* enable the DRAM PLL */
+       mmio_setbits_32(IMX_ANAMIX_BASE + 0x50, BIT(9));
+
        /* wait dram pll locked */
        while(!(mmio_read_32(DRAM_PLL_CTRL) & BIT(31)))
```

```
                    ;
```

5.5.4 Disable Arm PLL

You can disable Arm PLL in ATF to save power.

AHB_ROOT and the M core must use the same clock root, for example, a 24 MHz crystal or syspll1; otherwise, the system cannot wake up.

Code example:

Disable Arm PLL in ATF.

```
NOTICE("bypass ARM \n")
; /* set the a53 clk root 30388000 as 0x10000000, clk from 24M
 */
mmio write 32(IMX CCM BASE + 0x8000, 0x10000000);
/* set the a53 clk change to a53 clk root from ARM PLL */
mmio write 32(IMX CCM BASE + 0x9880, 0x00000000);
NOTICE("disable arm pll\n");
/* disable the ARM PLL, bypass first, then disable */
mmio setbits 32(IMX ANAMIX BASE + 0x84, BIT(4));
mmio_clrbits_32(IMX_ANAMIX_BASE + 0x84, BIT(9));Disable SYSPLL1
```

Resume Arm PLL back in ATF.

```
NOTICE("restore armpll\n");
/* enable the ARM PLL, enable first, then unbypass */
mmio_setbits_32(IMX_ANAMIX_BASE + 0x84, BIT(9));
while (!(mmio_read_32(IMX_ANAMIX_BASE + 0x84) & BIT(31)))
    ;
mmio_clrbits_32(IMX_ANAMIX_BASE + 0x84, BIT(4));
/* set the a53 clk root 30388000 as 0x10000000, clk from
 syspll1 800M */
mmio_write_32(IMX_CCM_BASE + 0x8000, 0x14000000);
/* set the a53 clk change to ARM PLL from a53 clk root */
mmio_write_32(IMX_CCM_BASE + 0x9880, 0x01000000);
NOTICE("ARM changed to ARM PLL \n");
```

5.5.5 Disable System PLLs

On i.MX 8M, there are three system PLLs, SYSPLL1, SYSPLL2, and SYSPLL3.

To disable system PLLs to save power, see the instructions below:

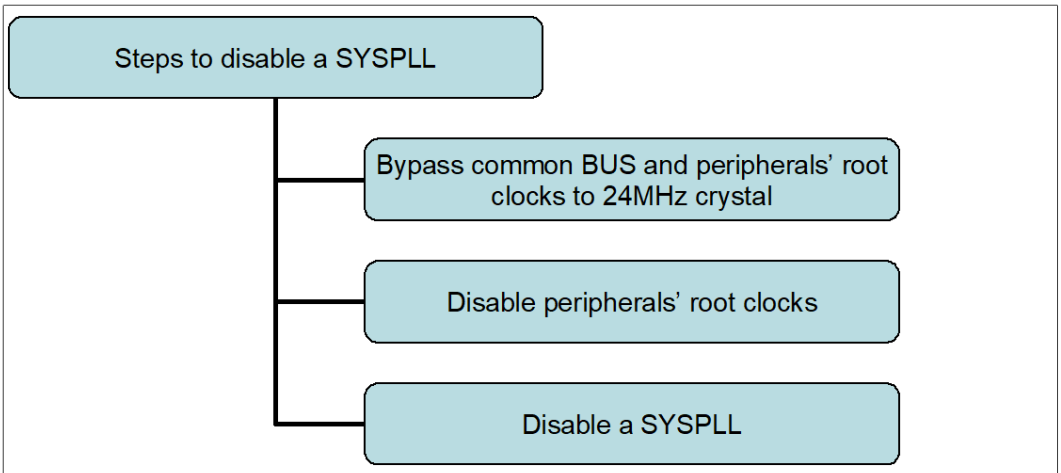

Figure 14. Steps to disable a SYSPLL

Before disabling SYSPLL, no modules' root clock must be using this SYSPLL; otherwise, the attempt to disable the SYSPLL leads to a system crash.

Common bus clocks in this case stand for main AXI, AHB, NOC.

5.5.5.1 Check modules' root clocks in the kernel

On i.MX 8M, modules' root clock can be configured in CCM_TARGET_ROOTn. You can learn modules' clock tree diagram from the kernel command.

```
cat /sys/kernel/debug/clk/clk_summary
```

On i.MX 8M, SYSPLL1 is the most widely used as modules' root clock. So pay attention to disabling SYSPLL1.

5.5.5.2 Bypass modules' root clocks on the SYSPLL to 24 MHz crystal

Those modules that must be bypassed are those clocks that must remain ON in low-power mode, and peripherals' clocks that are needed for the system, for example, GIC, and the M core application, for example, UART.

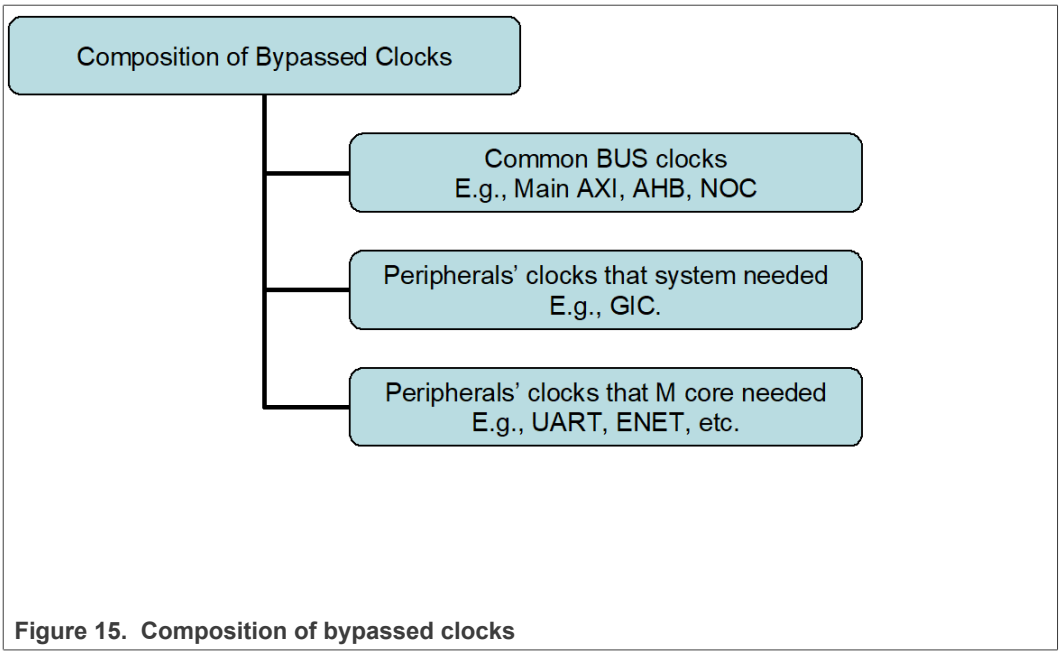

Figure 15. Composition of bypassed clocks

AHB_ROOT and the M core must use the same clock root, for example, a 24 MHz crystal or a syspll. Otherwise, the system may not be able to wake up.

DRoll back the change if the bypass of one clock root causes issues.

For example, on i.MX 8MP enet demo, bypassed root clocks are composed of common bus clocks (main AXI, AHB, NOC, and so on), GIC (system needed peripherals), ENET, and UART peripheral clocks (the M core application is needed).

Code example (for the enet low-power demo and system pll1 and pll2):

```
static save_root_regs syspll1_clk_root_bus_to_24m_registers[] =
 {
 {16, 0}, /*MAIN_AXI*/
 {17, 0}, /*ENET_AXI*/
 {26, 0}, /*NOC*/
 {27, 0}, /*NOC_IO*/ /*can't be touched! */
 {32, 0}, /*AHB_ROOT */ /* when disabling ARM PLL (bypass ARM
 PLL to 24MHz), this need to be bypassed to 24MHz crystal also.
 */
 {83, 0}, /*ENETREF*/
 {85, 0}, /*ENETPHY*/
 {94, 0}, /*UART1*/
};
static save_root_regs syspll2_clk_root_bus_to_24m_registers[] =
 {
    {100,0}, /*GIC*/
};
```

5.5.5.3 Disable modules' root clocks on the SYSPLL

Generally, unused modules' root clocks must be disabled to save power.

However, some peripherals' clocks might be hard to disable.

In that case, pay attention to the parent clocks of the peripheral clock. Check if the clock can be disabled with its parent clocks enabled or disabled.

Code example (for system pll1):

```
static root_clk_regs_save syspll1_clk_root_disable_registers[]
 = {
    {2, 0}, /*ML */
    {3, 0}, /*GPU3d*/
    {4, 0}, /*GPU SHADER*/
    {5, 0}, /*GPU 2D*/
    {6, 0}, /* AUDIO_AXI */
    {7, 0}, /*HSIO */
… ..
    {121, 0}, /*USDHC3*/
    {122, 0}, /*MEDIA_CAM1*/
    {125, 0}, /*MEDIA_CAM2*/
    {130, 0}, /*MEDIA_MIPI_TEST*/
    {131, 0}, /*ESPI3*/
    {134, 0}, /* SAI7*/
};
static root_clk_regs_save syspll2_clk_root_disable_registers[]
 = {
};
```

5.5.5.4 Disable the SYSPLL

If all root clocks that use the SYSPLL are bypassed or disabled, you can disable this SYSPLL.

Code example:

Disable SYSPLL2 in ATF.

```
+          syspll2_save   = mmio_read_32(IMX_ANAMIX_BASE +
 0x104);
+          NOTICE("disable syspll2\n");
+          /* disable the SYSTEM PLL2, bypass first, then
 disable */
+          mmio_setbits_32(IMX_ANAMIX_BASE + 0x104, BIT(4));
+          mmio_clrbits_32(IMX_ANAMIX_BASE + 0x104, BIT(9));
```

Restore SYSPLL2.

```
+          NOTICE("restore syspll2\n");
+          if (syspll2_save & BIT(9)) {
+                  /* enable the SYSTEM PLL2, enable first,
 then unbypass */
+                  mmio_setbits_32(IMX_ANAMIX_BASE + 0x104,
 BIT(9));
+                  while (!(mmio_read_32(IMX_ANAMIX_BASE +
 0x104) & (BIT(31))))
+                          ;
+                  mmio_clrbits_32(IMX_ANAMIX_BASE + 0x104,
 BIT(4));
+                  NOTICE("enabled SYSPLL2 \n");
+          }
```

5.5.5.5 Debugging

In case you failed to disable a system PLL, use these debugging tips.

1. Roll back the change and enable all bypassed clocks and disabled clocks. Check if the system can restore working.
2. Check if the clocks of some peripheral devices that use this SYSPLL are missed.
3. Add bypassed disabled clocks half by half and check if a clock operation causes the issue.

5.5.6 Disable unused CCGRs

The default value of the register in CCM_CCGR is 0x2. It means that for Domain 0 (A53 Domain), domain clocks are needed when in Run and Wait mode.

Keeping clocks ON in Wait mode consumes power. Change the value to 1, which means that domain clocks are only needed in Run mode.

It saves about 10 mW of power.

In this optimization, define CCGRs that must be reserved in low-power mode.

The CCGRs that must be reserved are the ones needed by the system and the application.

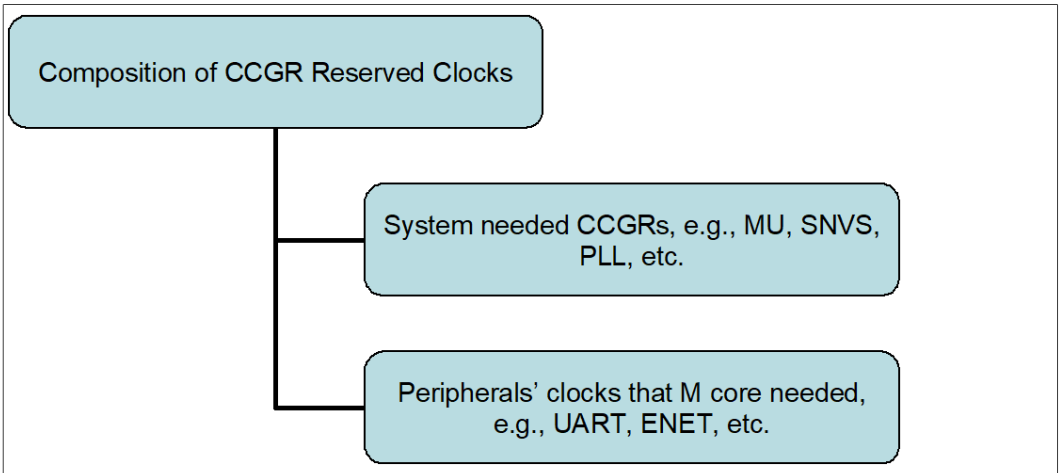

Figure 16. Composition of CCGR reserved clocks

Code example:

```
static ccgr_regs_save ccgr_disabled_registers[103];
static uint8_t ccgr_reserved_registers[] = {
    10, /* ENET1 */
    22, /* HS*/
    33, /* MU */
    36, /* OCRAM_S */
    64, /* SIM_ENET */
    71, /* SNVS */
    97, /* PLL */
};
… …
        NOTICE("Disable CCGR \n");
        for (uint32_t index = 0; index <
 ARRAY_SIZE(ccgr_reserved_registers); index++) {
```

```
ccgr_disabled_registers[ccgr_reserved_registers[index]].reserved
= 1;
        }
        for (uint32_t index = 0; index <
ARRAY_SIZE(ccgr_disabled_registers); index++) {
            if (ccgr_disabled_registers[index].reserved != 1) {
                ccgr disabled registers[index].value = mmio
read 32(IMX CCM BASE + 0x4000 +
16 * index) & 0xFF;
                if ((ccgr disabled registers[index].value !=
1) &&
(ccgr_disabled_registers[index].value != 0)) {
                    NOTICE("CCGR %d not 0 and 1\n", index);
                }
                mmio_write_32(IMX_CCM_BASE + 0x4000 + 16 *
index, 1);
            }
         }
```

5.5.7 Disable Audio PLL

If Audio PLLs are not used, you can disable them in ATF to save power.

In testing, disabling aplls might not have a benefit on VDD_SOC, but can reduce power on PLL_ANA_0V8.

Make sure that no module in root clocks uses APLLs, otherwise the attempt to disable leads to a crash.

Code example:

Disable APLLs in ATF.

```
+static uint8_t apll1enabled, apll2enabled;
+            /* Disabling apll1 and apll2 seems increase 15mW,
 230 => 245 */
+            apll1enabled = !!(mmio_read_32(IMX_ANAMIX_BASE) &
 BIT(31));
+            apll2enabled = !!(mmio_read_32(IMX_ANAMIX_BASE +
 0x14) & BIT(31));
+            if (apll1enabled) {
+                  mmio_setbits_32(IMX_ANAMIX_BASE, BIT(4));
+                  mmio_clrbits_32(IMX_ANAMIX_BASE, BIT(9));
+            }
+            if (apll2enabled) {
+                 mmio_setbits_32(IMX_ANAMIX_BASE + 0x14,
 BIT(4));
+                  mmio_clrbits_32(IMX_ANAMIX_BASE + 0x14,
 BIT(9));
+            }
```

Restore APLLs.

```
+            NOTICE("restore audiopll\n");
+            if (apll1enabled) {
+                  mmio_setbits_32(IMX_ANAMIX_BASE, BIT(9));
+                  while (!(mmio_read_32(IMX_ANAMIX_BASE) &
 (uint32_t)BIT(31)))
```

```
+                           ;
+                   mmio_clrbits_32(IMX_ANAMIX_BASE, BIT(4));
+           }
+           if (apll2enabled) {
+                   mmio_setbits_32(IMX_ANAMIX_BASE + 0x14,
 BIT(9));
+                   while (!(mmio_read_32(IMX_ANAMIX_BASE +
 0x14) & (uint32_t)BIT(31)))
+                           ;
+                   mmio_clrbits_32(IMX_ANAMIX_BASE + 0x14,
 BIT(4));
+           }
```

6 Debugging methods

Low-power applications need much effort in debugging, you can find several approaches here.

6.1 Enabling log prints in the kernel and ATF

Enable log prints in the kernel and ATF.

Table 11. Enabling log prints in the kernel and ATF

Item	Description
Kernel	echo N > /sys/module/printk/parameters/console_suspend
ATF	1. Find function bl31_early_platform_setup2 in <soc_name>_bl31_setup.c, for example, imx8mn_bl31_setup.c. 2. Change *console_set_scope(&console.console,CONSOLE_FLAG_BOOT);* to *console_set_scope(&console.console,CONSOLE_FLAG_RUNTIME);*

6.2 The system cannot wake up

1. Enable log to find out whether the system really cannot wake up or it crashes when entering Suspend mode.
2. Restore to a state that can wake up and add code changes one by one to see the root cause.
3. Check if `SLPCR_A53_FASTWUP_STOP_MODE` bit is set in SLPCR of GPC module. Generally, it should be done in function `imx_set_sys_lpm() of plat/imx/imx8m/gpc_common.c` in ATF code:

```
void imx set sys lpm(unsigned int last core, bool retention)
{
        if (retention)
            mmio clrsetbits 32(IMX GPC BASE + SLPCR, SLPCR
 A53 FASTWUP STOP MODE,
SLPCR EN DSM | SLPCR VSTBY | SLPCR SBYOS | SLPCR BYPASS PMIC
 READY);
        else
            mmio_clrsetbits_32(IMX_GPC_BASE + SLPCR,
 SLPCR_EN_DSM | SLPCR_VSTBY |
SLPCR SBYOS | SLPCR BYPASS PMIC READY, SLPCR A53 FASTWUP STOP
 MODE);
        /* mask M4 DSM trigger if M4 is NOT enabled */
        if (!imx is m4 enabled())
```

```
                mmio setbits 32(IMX GPC BASE + LPCR M4, BIT(31));
  … …
```

4. Check if `AHB_ROOT` and the M core are not using the same clock root, for example, a 24 MHz crystal or syspll1.

7 Power measurement results

This section gives a brief overview of the power measurement results.

7.1 Power measurement tool

This paper uses a power board to do the measurements. Two tools are shown below.

Table 12. Tools to do the measurements

Name	GITHUB URL	Usage
BCU	https://github.com/NXPmicro/bcu	Command: *>> ./bcu monitor -board=<board_name>* Example on imx8mp: *>> ./bcu monitor -board=imx8mpevkpwra0* For detail usage, check BCU.pdf in the BCU release package.
PMT	https://github.com/NXPmicro/pmt	Command: *>> python main.py eeprom -m write -f docs/EEPROM_Programmer_Tool.yaml* *>> python main.py monitor -m gui* For details, see AN13119 on nxp.com.

7.2 Power test results

The data presented in this application note is based on empirical measurements taken on a small sample, so the presented results are not guaranteed.

7.2.1 Power results for the A core in Stop mode

Tested with low-power demo on i.MX 8MP power board with the M core in three modes, Run, Stop, and Wait.

7.2.1.1 Put the A core into Suspend mode and the M core into Run

In this test, the A core is suspended and the M core is running at 24 MHz. No optimization code in this case.

The power of VDD_SOC is 29.5 mW.

	Voltage(V)				Current(mA)/(uA)				Power(mWatt)/(uWatt)				Max Range Select(mA)	
location	now	avg	max	min	now	avg	max	min	now	avg	max	min	Range1	Range2
A vdd_arm	1.01	1.00	1.01	0.99	-0.6	0.2	3.4	-2.7	-0.6	0.2	3.4	-2.8	[*]5000.0	[ ]20.0
B nvcc_dram_1v1	1.10	1.10	1.11	1.09	-0.5	0.2	33.5	-3.2	-0.6	0.2	36.9	-3.5	[*]2000.0	[ ]10.0
C vsys_5v	5.25	5.24	5.25	5.23	69.3	78.7	220.2	24.9	363.5	412.8	1154.0	130.4	[*]5000.0	[ ]49.5
D vdd_soc	0.85	0.85	0.86	0.84	21.7	34.7	93.7	18.0	18.4	29.5	79.2	15.3	[*]10000.0	[ ]49.8
E lpd4_vddq	1.10	1.10	1.11	1.09	-0.1	-0.2	21.1	-3.0	-0.1	-0.2	23.2	-3.3	[*]2000.0	[ ]10.0

Figure 17. The A core is in Suspend mode and the M core is in Run

7.2.1.2 Put the A core into Suspend mode and the M core into Stop

In this test, the A core is suspended and the M core is put into Stop mode. The system is in DSM when the M core enters Stop mode. No optimization code in this case.

The power of VDD_SOC is 10.7 mW.

	Voltage(V)				Current(mA)/(uA)				Power(mWatt)/(uWatt)				Max Range Select(mA)	
location	now	avg	max	min	now	avg	max	min	now	avg	max	min	Range1	Range2
A vdd_arm	-0.00	0.00	0.01	-0.00	2.1	-0.1	2.6	-2.3	-0.0	-0.0	0.0	-0.0	[*]5000.0	[ ]20.0
B nvcc_dram_1v1	1.10	1.10	1.10	1.09	0.9	0.3	20.0	-3.5	1.0	0.4	22.0	-3.9	[*]2000.0	[ ]10.0
C vsys_5v	5.24	5.25	5.25	5.24	138.4	57.1	138.7	8.5	725.7	299.2	727.1	44.8	[*]5000.0	[ ]49.5
D vdd_soc	0.85	0.85	0.86	0.84	10.4	12.6	79.3	5.8	8.8	10.7	67.5	4.9	[*]10000.0	[ ]49.8

Figure 18. The A core is in Suspend mode and the M core is in Stop

7.2.1.3 Put the A core into Suspend mode and the M core into Wait

In this test, the A core is suspended and the M core is put into Wait mode. The system is not in DSM as the M core enters Wait mode. No modification code in this case.

The power of VDD_SOC is 16.9 mW.

	Voltage(V)				Current(mA)/(uA)				Power(mWatt)/(uWatt)				Max Range Select(mA)	
location	now	avg	max	min	now	avg	max	min	now	avg	max	min	Range1	Range2
A vdd_arm	1.00	1.00	1.01	0.99	2.1	0.8	29.9	-2.3	2.1	0.8	30.0	-2.3	[*]5000.0	[ ]20.0
B nvcc_dram_1v1	1.09	1.10	1.11	1.09	0.2	-0.1	0.9	-3.1	0.3	-0.2	1.0	-3.4	[*]2000.0	[ ]10.0
C vsys_5v	5.24	5.25	5.25	5.24	41.7	65.4	142.8	16.9	218.5	343.2	749.8	88.8	[*]5000.0	[ ]49.5
D vdd_soc	0.85	0.85	0.85	0.84	12.8	19.9	81.5	10.4	10.9	16.9	69.2	8.9	[*]10000.0	[ ]49.8
E lpd4_vddq	1.10	1.10	1.10	1.09	-0.1	-0.5	1.3	-2.4	-0.1	-0.5	1.5	-2.7	[*]2000.0	[ ]10.0

Figure 19. The A core is in Suspend mode and the M core is in Wait

7.2.2 Power optimization results with the A core in Fast-Wake-up-Stop mode

On i.MX 8MP power board, the result for enet demo case is below (A53 suspend while M7 is monitoring for syncing Ethernet packets).

Before optimization, the power of VDD_SOC is 897 mW:

	Voltage(V)				Current(mA)/(uA)				Power(mWatt)/(uWatt)				Max Range Select(mA)	
location	now	avg	max	min	now	avg	max	min	now	avg	max	min	Range1	Range2
A vdd_arm	0.85	0.85	0.86	0.84	8.2	10.6	95.5	3.5	7.0	9.0	81.2	3.0	[*]5000.0	[ ]20.0
B nvcc_dram_1v1	1.09	1.09	1.10	1.09	57.9	68.1	98.1	55.5	63.1	74.5	107.5	60.4	[*]2000.0	[ ]48.8
C vsys_5v	5.23	5.23	5.24	5.22	544.1	536.7	601.5	472.3	2847.3	2807.5	3146.6	2468.7	[*]5000.0	[ ]238.1
D vdd_soc	0.93	0.93	0.94	0.92	957.3	962.8	984.5	948.5	889.8	897.0	921.5	880.1	[*]10000.0	[ ]99.0

Figure 20. Results before optimization

After optimization, the power of VDD_SOC is 43.5 mW:

	Voltage(V)				Current(mA)/(uA)				Power(mWatt)/(uWatt)				Max Range Select(mA)	
location	now	avg	max	min	now	avg	max	min	now	avg	max	min	Range1	Range2
A vdd_arm	1.01	1.00	1.01	0.99	0.0	0.2	3.7	-2.1	0.0	0.2	3.7	-2.1	[*]5000.0	[ ]20.0
B nvcc_dram_1v1	1.10	1.10	1.11	1.09	0.1	0.4	36.7	-2.1	0.1	0.4	40.6	-2.4	[*]2000.0	[ ]10.0
C vsys_5v	5.24	5.24	5.25	5.23	39.8	80.6	219.1	26.2	208.7	422.5	1147.9	137.6	[*]5000.0	[ ]49.5
D vdd_soc	0.85	0.85	0.86	0.84	33.0	51.2	107.4	31.7	28.0	43.5	91.6	26.9	[*]10000.0	[ ]49.8

Figure 21. Results after optimization

Our optimization can largely improve the power on i.MX 8MP.

8 Example software

Software examples are low-power demo for standard release and Ethernet demo to demonstrate the optimization changes for ATF.

The demo code is based on the L5.4.47_2.2.0 kernel release.

Check the patch for details.

9 References

- *i.MX 8M Mini Heterogenous Low Power Voice Control Solution* (document: AN13201)
- *i.MX 8M Mini Power Consumption Measurement* (document: AN12410)
- *i.MX Power Measurement Tool* (document: AN13119)
- *How to Reduce SoC Power when Running M4 with A53on i.MX 8M* (document: AN12225)

10

Table 13. Revision history

Revision number	Date	Substantive changes
0	04 October 2021	Initial release
1	22 September 2022	Sections 5.5.7 and 8 are modified

11 Legal information

11.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

11.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

11.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Contents

For more information, please visit: http://www.nxp.com

Date of release: 22 September 2022
Document identifier: AN13400