AN14140

Keep GPIO State in Low Power – i.MX 8ULP

Rev. 1 — 4 December 2023 Application note

Document information

Information	Content
Keywords	AN14140, i.MX 8ULP, FSGPIO, RGPIO, GPIO state keep, uPower, Linux, SDK, low power modes, PAD isolation, CM33, A35
Abstract	This application note describes how to keep GPIO state in low power modes including the transition.

1 Overview

i.MX 8ULP I/O design provides the feature of keeping the GPIO state (output HIGH or LOW) in low-power modes and even during the power mode transition. This makes hardware board design easier when users want to use GPIO to control the external peripherals state. For example, using GPIO to control the Wi-Fi module power supply, keep its power during system suspend to enable Wake On WLAN to wakeup system. In this case, always power on the Wi-Fi module.

This documentation describes how to keep GPIO state in low-power modes including the transition.

2 i.MX 8ULP I/O design

Figure 1 shows the big picture of the i.MX8ULP I/O Design. The IOMUX cell is required whenever two or more functional modes are required for a specific pad. For example, the PTA0 port PAD can be used by General Purpose I/O (GPIO, PTA0), LPUART (`LPUART0_CTS_b`), or I^{2}C (`LPI2C0_SCL`). Through IOMUXC PCR register MUX bits, you can configure by which module the PAD is used. The PAD attributes, such as drive strength, slew rate, and pull up/down, are also configurable through PCR registers.

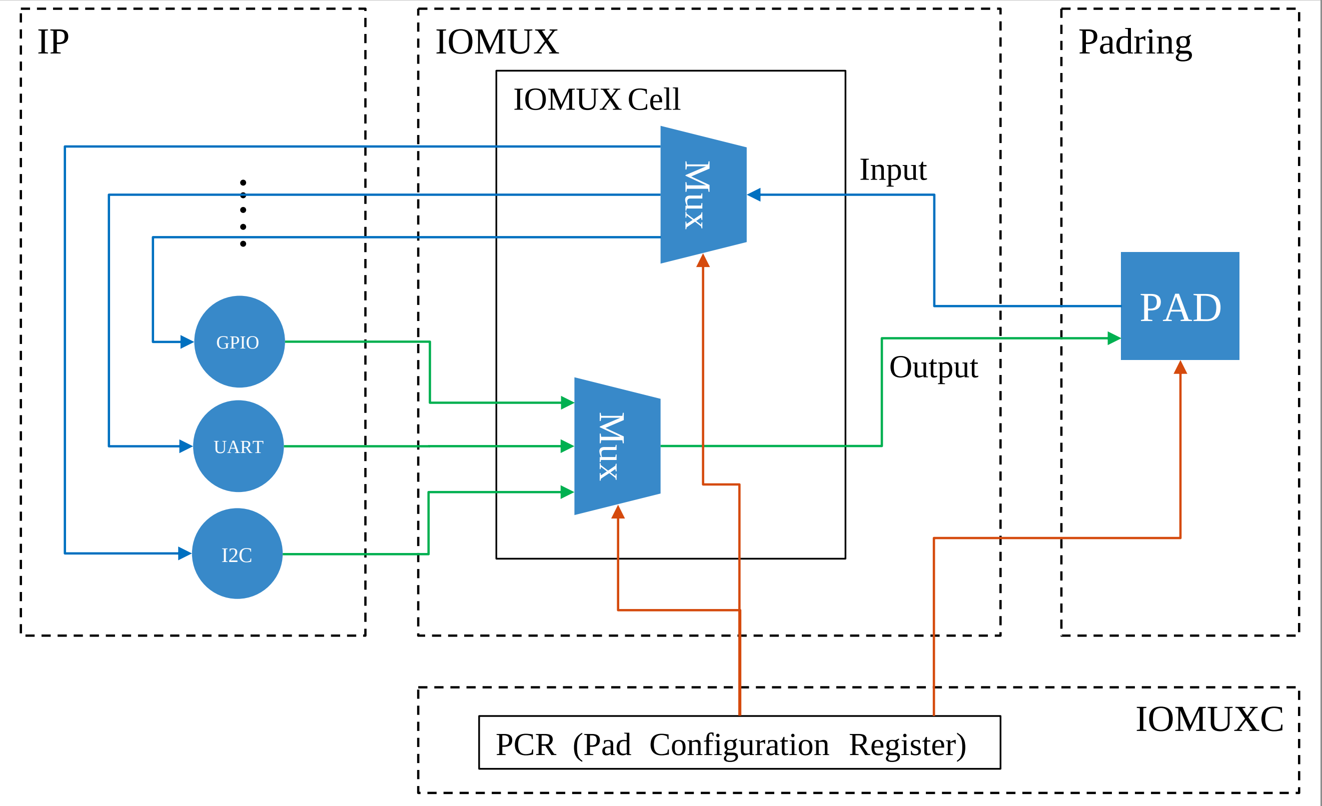

Figure 1. i.MX 8ULP I/O design

2.1 IOMUX design

The IOMUX Controller (IOMUXC), working with the IOMUX, enables the chip to share one pad for multiple signals from different peripheral interfaces. To perform this pad sharing mechanism, multiplex the input and output signals of the pad. Every peripheral signal requires a specific pad setting parameter, such as weak pull-up or drive-strength. The IOMUXC controls the pads setting parameters, digital filter functions, and peripherals multiplexing of the pad. For each pad, there are up to 16 peripherals multiplexing options, called ALT modes. The input signal of a peripheral may come from multiple pads. To avoid an input signal enabled at multiple locations, the IOMUXC also controls input multiplexing logic. The IOMUX consists of combinatorial logic combined from several basic IOMUX cells. Each basic IOMUX cell handles the multiplexing of one pad signal as defined on IOMUXC registers.

2.2 PAD design

i.MX 8ULP has two pad types:

- Failsafe GPIO (FSGPIO)
  - Operating voltage range: 1.8 V ~3.3V. The range is configurable.
  - Isolation: close - keeps the state of the pad before turning off the domain.
  - Isolation: reset - resets level shift and weak drive.
  - Used by PTA/B/E/F.
- High Speed GPIO (HSGPIO)
  - Operation voltage range: 1.8 V ~3.3V. The range is configurable.
  - Automatic compensation: adjusts drive strength dynamically; freeze stops compensation uses last value (lowest power).
  - When the power supply is off,the pads go to the high Z state. The state is not retained.
  - Used by PTC/D.

In this case, FSGPIO PAD is recommended, because it can do isolation to keep its state retention even when IOMUX and GPIO power gated (IP not functional). It means that for this purpose, use PTA/B (RTD domain) or PTE/F (APD domain).

3 Keep GPIO state in power down

To keep GPIO state in power down modes, enable the FSGPIO PAD isolation function.

The i.MX 8ULP IP modules have their own statuses in different power modules. Table 1 describes the example for IOMUX and GPIO modules in real-time domain.

Table 1. IOMUX and GPIO modules in real-time domain

Modules	**Power modes**	**Active**	**Sleep**	**Deep sleep**	**Power down**
	Power state power domain	**Core supply = ON, Bias = AFBB and DVS, System/ Bus clocks = ON, I/O supply = ON**	**Core Supply =ON, Bias = AFBB or ARBB, Voltage = Fixed, System/ Bus clock = ON (optional) IO supply = ON**	**Core supply = ON, Bias = RBB Voltage/Bias = prog, System/Bus clock = OFF, I/O supply = ON**	**Core supply = ON (Mem only), Bias = RBB, Voltage/ Bias = prog, System/Bus clock = OFF, I/O supply = ON (optional)**
IOMUX [0]	RTD	Functional	CG or Static (optional)[1]	CG	PG
RGPIOA, B, C	RTD	Functional	CG (optional)[2]	CG	PG

[1] If the system/bus clock is not available (OFF), the module is clock gated. Otherwise, it remains static.
[2] If the system clock is OFF in the sleep mode, the module is clock gated. Otherwise, it is fully functional.

The RTD IOMUX and GPIO can be Clock Gated (CG) optionally under the sleep mode, CG under the Deep Sleep mode, and Power Gated (PG) under the power-down mode. To keep the external I/O pin state in the sleep or deep sleep mode, keep the IOMUX and GPIO configurations of the pin unchanged, because GPIO state can be kept even when the clock is gated. But under the power down mode, user can only use the PAD isolation feature.

To use PAD isolation to keep GPIO state, a dedicated programing flow is needed. In brief, make sure to enable the PAD isolation before IOMUX/GPIO clock or power gated. And disable the isolation after IOMUX/GPIO registers are restored:

1. Configure IOMUX and GPIO to make the pin output to the required level.
2. Keep and save IOMUX and GPIO configuration for the pin before entering the low-power mode.
3. Enable FSGPIO PAD isolation by uPower before entering the low-power mode.
4. Keep FSGPIO PAD isolation enabled even when exiting from the low-power mode by uPower.
5. Restore IOMUX and GPIO configuration for the pins after exiting from the low-power mode.
6. Disable FSGPIO PAD isolation.

The programing flows for APD and RTD domains are a little bit different. They need two sections to cover. Meanwhile, PTE or PTF is used in APD in this case and PTA or PTB is used in RTD for this case.

3.1 Application Domain (APD)

This section uses the Linux BSP as an example to describe how to configure IOMUX/GPIO/PAD for APD.

3.1.1 Pin configuration and usage

Table 2 describes two ways to configure and use GPIO in Linux.

Table 2. Configure and use GPIO

	General GPIO device driver	Dedicated device driver
Configure	GPIO `pinctrl` hogging in device tree	Device driver `pinctrl` in device tree.
Usage	`gpiod` utility in user space	Handle in driver, like toggling GPIO during driver probe, to reset or power up the device.

- General GPIO device driver.
  User can use GPIO hogging in the device tree to configure a PAD pin to GPIO mux and its PAD attributes. GPIO hogging is a mechanism providing automatic GPIO request and configuration as part of the driver probe function of the gpio-controller.

```
&iomuxc {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_hog>;
        pinctrl_hog: hoggrp {
                fsl,pins = <
                        MX8ULP_PAD_PTF26__PTF26 0x3 // 0x3 is PAD attributes
 Pull-up selected
                >;
        };
......
```

  After boot, users can use `gpiod` utility to toggle GPIO. Taking PTF26 as an example:
  Set the PTF26 output LOW:

```
$gpioset -c 5 26=0
```

  Set the PTF26 output HIGH:

```
$gpioset -c 5 26=1
```

- Dedicated driver
  User can use `pinctrl` nodes in the device tree to configure a PAD pin to GPIO mux and its PAD attributes. Taking SDIO driver (connect external WiFi module) as an example, `mmc-pwrseq-simple` driver takes one

property `reset-gpios` to use this pin as power switch for connected Wi-Fi module power. To initialize this pin setting during device driver probe, assign `pinctrl_usdhc2_ptf` to `pinctrl` of the `usdhc2` device node. And assign PTF26 to `reset-gpios` in `usdhc2_pwrseq` node as below:

```
&iomux1 {
    pinctrl_usdhc2_ptf: usdhc2ptfgrp {
        fsl,pins = <
            MX8ULP_PAD_PTF26__PTF26         0x3
        >;
    };
    …
};

usdhc2_pwrseq: usdhc2_pwrseq {
    compatible = "mmc-pwrseq-simple";
    reset-gpios = <&gpiof 26 GPIO_ACTIVE_LOW>;
};

&usdhc2 {
        pinctrl-names = "default", "state_100mhz", "state_200mhz", "sleep";
        pinctrl-0 = <&pinctrl_usdhc2_pte>, <&pinctrl_usdhc2_ptf>;
        pinctrl-1 = <&pinctrl_usdhc2_pte>, <&pinctrl_usdhc2_ptf>;
        pinctrl-2 = <&pinctrl_usdhc2_pte>, <&pinctrl_usdhc2_ptf>;
        pinctrl-3 = <&pinctrl_usdhc2_pte>, <&pinctrl_usdhc2_ptf>;
        mmc-pwrseq = <&usdhc2_pwrseq>;
…
};
```

3.1.2 Program flow

Arm Trusted Firmware (AFT) deals with the program flow for keeping the GPIO state in low-power modes. To modify the ATF source code:

1. Do not reset dedicated IOMUX/GPIO registers. Take PTF26 as an example (*plat/imx/imx8ulp/apd_context.c*):

```
void apd_io_pad_off(void) {
……
        /* off the PTD/E/F, need to be customized based on actual user case
 */
        for (i = 0; i < 3; i++) {
                for (j = 0; j < iomuxc_sections[i].reg_num; j++) {
-                       mmio_write_32(iomuxc_sections[i].offset + j * 4, 0);
+                       if (!(i == 2 && j == 26))
+                               mmio_write_32(iomuxc_sections[i].offset + j *
 4, 0);
                }
        }
 ……
}
```

2. Configure `PAD_CFG` to **CLOSED** for PTE/F for APD Power Down.
   Configure (keep) `PAD_CFG` (in uPower structure array by `ADMA_PWR_MODE` index) as **CLOSED** after resume to **ACTIVE** (*plat/imx/imx8ulp/imx8ulp_psci.c*).

```
ps_apd_pwr_mode_cfgs_t apd_pwr_mode_cfgs = {
 ……
        /* PD */
        [PD_PWR_MODE] = {
                .swt_board_offs = 0x170,
                .swt_mem_offs = 0x178,
                .pmic_cfg = PMIC_CFG(0x23, 0x0, 0x2),
                .pad_cfg = PAD_CFG(0xC, 0x0, 0x01e80a00),
                .bias_cfg = BIAS_CFG(0x0, 0x2, 0x2, 0x0),
        },

        [ADMA_PWR_MODE] = {
                .swt_board_offs = 0x120,
                .swt_mem_offs = 0x128,
                .pmic_cfg = PMIC_CFG(0x23, 0x0, 0x2),
                .pad_cfg = PAD_CFG(0xC, 0x0, 0x0deb7a00),
                .bias_cfg = BIAS_CFG(0x2, 0x2, 0x2, 0x0),
        },
……
};
```

The `pad_close` member in the `pad_cfg` highlight in GREEN color here is to tell uPower to close which FSGPIO PAD: Bit 0: PTA, Bit 1: PTB, Bit 2: PTE, Bit 3: PTF. So, here `0xC` means to enable isolation for PTE and PTF.

3. Restore IOMUX/GPIO registers after resume to ACTIVE. By default,ATF handles the restore, so no change is required.
4. Clear the bits of `PMC_SYS_CTRL_PAD[PADCLOSE]` to remove isolation of PADs after IOMUX/GPIO register is restored after resume (*plat/imx/imx8ulp/imx8ulp_psci.c*).

```
 #define DRAM_LPM_STATUS                U(0x2802b004)
+#define PMC_SYS_CTRL_PAD       U(0x283590BC)
 extern void imx8ulp_init_scmi_server(void);
void imx_domain_suspend_finish(const psci_power_state_t *target_state)
                /* restore the ap domain context */
                imx_apd_ctx_restore(cpu);
+               /* restore the PAD status */
+               mmio_write_32(PMC_SYS_CTRL_PAD, 0x0);
  ……
}
```

3.2 Real-Time Domain (RTD)

This section uses MCUXpresso SDK as an example to describe how to configure IOMUX/GPIO/PAD for RTD.

3.2.1 Pin configuration and usage

In SDK, to configure IOMUX, refer to *pin_mux.c* in the demo applications. For example:

```
void BOARD_InitXXPins(void) {
    IOMUXC_SetPinMux(IOMUXC_PTA4_PTA4, 0U); // Configure PTA4 MUX to GPIOA pin
 4.
    IOMUXC_SetPinConfig(IOMUXC_PTA4_PTA4, // Configure it's PAD configure to
 Pull-up
IOMUXC_PCR_PE_MASK | IOMUXC_PCR_PS_MASK);
}
```

GPIO output can be configured in application. For example:

```
rgpio_pin_config_t gpioConfig = {
        kRGPIO_DigitalOutput,
        0U,
};
```

```
RGPIO_PinInit(GPIOA, 4, &gpioConfig); // Configure PTA4: GPIOA pin4 as output
RGPIO_PinWrite(GPIOA, 4, 1); // Output HIGH
```

3.2.2 Program flow

In the SDK (taking SDK v2.14.0 as example), the low power configuration flow is done in the application: take `power_mode_switch` demo as example to keep PTA4 state during the low-power mode:

1. Do not reset dedicated IOMUX/GPIO registers (*power_mode_switch.c*).

```
static void APP_Suspend(void)
{
 ……
    /* Backup PTA IOMUXC and GPIOA ICR registers then disable */
    for (i = 0; i <= 24; i++)
    {
        iomuxBackup[backupIndex] = IOMUXC0->PCR0_IOMUXCARRAY0[i];
        gpioICRBackup[backupIndex] = GPIOA->ICR[i];
        GPIOA->ICR[i] = 0; /* disable interrupts */
if (i != 4) // PTA4
            IOMUXC0->PCR0_IOMUXCARRAY0[i] = 0;
        backupIndex++;
    }
```

2. Configure `PAD_CFG` to **CLOSED** for PTA/B when entering RTD Power Down. Configure (keep) `PAD_CFG` (in `uPower` structure array by `ACT_RTD_PWR_MODE` index) as CLOSED after resume to ACTIVE (*lpm.c*):

```
static ps_rtd_pwr_mode_cfgs_t rtd_pwr_mode_cfgs = {
    /* PD */
    [PD_RTD_PWR_MODE] =
        {
            .in_reg_cfg     = IN_REG_CFG(0x00000000, 0x00000000),
            .pmic_cfg       = PMIC_CFG(0x00000023, 0x00000000),
            .pad_cfg        = PAD_CFG(0x3, 0x00000000, 0x00000000),
            .mon_cfg        = MON_CFG(0x00000000, 0x0, 0x0),
            .bias_cfg       = BIAS_CFG(0x00010001, 0x0001001a, 0x0001001a,
 0x00000001),
            .pwrsys_lpm_cfg = PWRSYS_LPM_CFG(0),
        },
 ……
    /* ACT */
    [ACT_RTD_PWR_MODE] =
        {
            .in_reg_cfg     = IN_REG_CFG(0x0000001c, 0x3),
            .pmic_cfg       = PMIC_CFG(0x23, 0x00000000),
            .pad_cfg        = PAD_CFG(0x3, 0x0, 0x0),
            .mon_cfg        = MON_CFG(0x0, 0x0, 0x0),
            .bias_cfg       = BIAS_CFG(0x00020002, 0x2, 0x2, 0x0),
            .pwrsys_lpm_cfg = PWRSYS_LPM_CFG(0),
        },
};
```

The `pad_close` member in the `pad_cfg` highlight in GREEN color here is to tell uPower to close which FSGPIO PAD: Bit 0: PTA, Bit 1: PTB, Bit 2: PTE, Bit 3: PTF. So, here `0x3` means to enable isolation for PTA and PTB.

3. Restore IOMUX/GPIO registers after resumed to **ACTIVE**. By default,it is restored in the `APP_Resume()`:*power_mode_switch.c*.
4. Clear the bits of `PMC_SYS_CTRL_PAD[PADCLOSE]` to remove isolation of PADs.

```
static void APP_Resume(bool resume)
```

```
{
 ……
    /* Restore PTC IOMUXC and GPIOC ICR registers */
    for (i = 0; i <= 23; i++)
    {
        IOMUXC0->PCR0_IOMUXCARRAY2[i] = iomuxBackup[backupIndex];
        GPIOC->ICR[i]                 = gpioICRBackup[backupIndex];
        backupIndex++;
    }
*(volatile uint32_t *)0x283590BC = 0x0;
……
}
```

4 Reference

- *MX8ULP Reference Manual*
- *uPower User Guide* (document UPOWERFWUG)
- *i.MX Linux Reference Manual* (document IMXLXRM)
- SDK demo

5 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

6 Revision history

Table 3 summarizes the revisions to this document.

Table 3. Revision history

Revision number	Release date	Description
1	04 December 2023	Initial public release

Contents

Date of release: 4 December 2023
Document identifier: AN14140