

Table of Contents

[Table of Contents](#)

[Overview](#)

[References](#)

[Power management on the i.MX53 ARD board](#)

[Low-level Power Management \(PM\) Driver](#)

[Hardware Operation](#)

[Software Operation](#)

[Kernel configuration](#)

[Code Analysis](#)

[system.c](#)

[pm.c](#)

[Dynamic Voltage and Frequency Scaling Core \(DVFSC\)](#)

[Hardware Operation](#)

[Software Operation](#)

[Kernel configuration](#)

[Code Analysis](#)

[dvfs_core.c](#)

[Software Interface](#)

[CPU Frequency Scaling \(CPUFREQ\) Driver](#)

[Software Operation](#)

[Kernel configuration](#)

[Software Interface](#)

[Software Based Peripheral Domain Frequency Scaling](#)

[Software Interface](#)

[Kernel configuration](#)

[Software Operation](#)

[Study of the LTC3589 driver](#)

[Kernel configuration](#)

[Source files](#)

[Using the driver](#)

Overview

This document is intended to provide an understanding of the Linux Power Management features of the LTC3589 PMIC driver. In particular the focus is on Freescale iMX5x platforms as this is the basis for the current driver implementation.

The Linux Kernel referenced is version 2.6.35 with modifications to support Freescale's

implementation of the Android 2.3 (Gingerbread). The code was provided from Freescale's Android Release R10 freely available on their website. The link is available by visiting [iMX53 Board Support](#) and then selecting [i.MX53 Android 10 Source Code](#). Registration may be required.

References

- i.MX53 START Linux Reference Manual, Rev.11.01.00 (01/2011)
- i.MX53 Multimedia Applications Processor Reference Manual
- LTC3589 Reference Manual

Power management on the i.MX53 ARD board

The software implementation of power management for the i.MX53 ARD board is described in chapters 16-17-18-19 of the Linux reference manual (although it is not the manual for the ARD board, the information that it provides is relevant for all i.MX53 implementations).

The main subsystems are:

- Low-level Power Management (PM) Driver: controls low-power modes (e.g. System idle, Standby...)
- Dynamic Voltage Frequency Scaling (DVFS) Driver: Controls dynamic modifications of the frequency and voltage of the CPU core power domain.
- Software Based Peripheral Domain Frequency Scaling: Dynamically changes the clocks in the peripheral domain.
- CPU Frequency Scaling (CPUFREQ) driver: Implements the CPU frequency changes and working points, which are a combination of the {CPU voltage, CPU frequency, PLL rate}.

Low-level Power Management (PM) Driver

Most of the information in this paragraph has been copied from the Linux Reference Manual - Chapter 16. Please refer to this document for more information.

Hardware Operation

The i.MX5 supports four low power modes: RUN, WAIT, STOP, and LPSR (low power screen).

Mode	Core	Modules	PLL	CKIH/FPM	CKIL
RUN	Active	Active, Idle or Disable	On	On	On

WAIT	Disable	Active, Idle or Disable	On	On	On
STOP	Disable	Disable	Off	Off	On
LPSR	Disable	Disable	Off	On	On

Software Operation

The i.MX5 PM driver maps the low-power modes to the kernel power management states as listed below:

- *Standby*: maps to WAIT mode which offers minimal power saving, while providing a very low-latency transition back to a working system
- *Mem* (suspend to RAM): maps to STOP mode which offers significant power saving as all blocks in the system are put into a low-power state, except for memory, which is placed in self-refresh mode to retain its contents
- *System idle*—maps to WAIT mode

Kernel configuration

- **CONFIG_PM**: Build support for power management. In *menuconfig*, this option is available under *Power management options > Power Management support*. By default, this option is Y.
- **CONFIG_SUSPEND**: Build support for suspend. In *menuconfig*, this option is available under *Power management options > Suspend to RAM and standby*

Code Analysis

The code implementing the PM driver is mainly located in the two following files - available in *arch/arm/mach-mx5*:

File	Description
pm.c	Supports suspend operation
system.c	Supports low-power modes

Note: The documentation also mentions *wfi.S* and *suspend.S*, but these files are generic to the i.MX5x SoCs and do not involve the PMIC. We will only focus on the areas where the PMIC is involved.

system.c

This file contains *arch_idle()* and *mxc_cpu_lp_set()*, which set the CPU low-power mode before issuing the wait for interrupt instruction. This code takes care of settings the right clocks depending on the low-power mode that has been requested. It does not directly involve the PMIC.

pm.c

Implements the *Standby* and *Mem* states. This file is also generic to the i.MX5x but has some hooks to another PMIC driver (DA9053 - used on the i.MX53 Quick Start Board) which are not required for the LTC3589.

mx5_suspend_prepare() is called before entering the suspend state (implemented in *mx5_suspend_enter()* which also calls the functions in *system.c*). It sets the CPU working point to 400MHz (workaround for the i.MX53) which causes PMIC driver to adjust the voltage for the CPU (more details in the next section).

Dynamic Voltage and Frequency Scaling Core (DVFS)

Most of the information in this paragraph has been copied from the Linux Reference Manual - Chapter 17. Please refer to this document for more information.

Hardware Operation

The DVFS allows simple dynamic voltage frequency scaling:

- The frequency of the core clock domain and the voltage of the core power domain can be changed on the fly while all blocks (including the ARM platform) continue their normal operation. The frequency of the core clock domain can be changed by switching temporarily to an alternate PLL clock, and then get back to the updated PLL, already locked at a specific frequency, or by merely changing the post dividers division factors.
- DVFS is a monitor that only provides an interrupt when counting exceeds a predefined value (e.g. the CPU load is high or low) and does not actually send request to make a change a change of voltage and frequency. The interrupt is processed in software by the DVFS driver.
- The voltage of the core power domain is changed through the PMIC.

For more information on the hardware DVFS Core block, refer to the DVFS chapter in the *Multimedia Applications Processor* documentation.

Software Operation

The DVFS device driver allows the frequency of the core clock domain and the voltage of the core power domain to be changed on the fly.

- The frequency of the core clock domain and the voltage of the core power domain are changed by switching between defined freq-voltage operating points.
- The frequencies are manipulated using the clock framework API. More information about the clock framework API: <http://www.kernel.org/doc/html/docs/kernel-api/clock.html>
- The voltage is set using the regulators API. More information in the Linux kernel documentation: <http://lxr.linux.no/#linux+v2.6.35.9/Documentation/power/regulator/>

Kernel configuration

There are no menu configuration options for this driver. The DVFS core is included by default.

Code Analysis

- The code implementing the DVFS Core driver is located in *arch/arm/plat-mxc/dvfs_core.c*
- For DVFS CPU working point settings, see *arch/arm/mach-mx5/mx53_wp.c*

dvfs_core.c

- The interrupts raised by the DVFS controller are handled in *dvfs_irq()*, which schedules a work item handled in *dvfs_core_work_handler()*. The CPU frequency is changed using *set_cpu_freq()*.
- The available CPU working points (frequencies/voltages) are defined in *arch/arm/mach-mx5/mx53_wp.c*:

- Different arrays corresponding to the different versions i.MX53 SoC (Automotive, Consumer 1GHz and 1.2GHz) define all the possible working points.
- The Automotive revision (IMX53_AEC) only has one working point defined by the following structure:

```
/* working point for auto*/
static struct cpu_wp cpu_wp_aec[] = {
    {
        .pll_rate = 800000000,
        .cpu_rate = 800000000,
        .pdf = 0,
        .mfi = 8,
        .mfd = 2,
        .mfn = 1,
        .cpu_podf = 0,
        .cpu_voltage = 1050000, },
};
```

This implies that the ARD board (which has the automotive version of the i.MX53) only has one working point (CPU at 800MHz).

Other revisions of the i.MX53 (e.g. the one present on the Quick Start Board) can scale the CPU frequency between 1.2GHz and 160MHz.

- *set_cpu_freq()* also calls *regulator_set_voltage(core_regulator, gp_volt, gp_volt)* to adjust the voltage of the core regulator. This voltage is set by the PMIC itself.
 - The handle to the regulator is obtained at initialization (i.e. boot time since the DVFS core driver is built-in) through the regulator API in *mxc_dvfs_core_probe()*:

```
core_regulator = regulator_get(NULL, dvfs_data->reg_id);
```
 - In order to make let the DVFS driver - which is generic with regards to the regulators used - know that this regulator (SW1 on the PMIC) is available on the ARD board, the
-

following structures are defined in *arch/arm/mach-mx5/mx53_ard.c*:

```
static struct mxc_dvfs_platform_data dvfs_core_data = {
    .reg_id = "SW1",
    .clk1_id = "cpu_clk",
    .clk2_id = "gpc_dvfs_clk",
    .gpc_cntr_offset = MXC_GPC_CNTR_OFFSET,
    .gpc_vcr_offset = MXC_GPC_VCR_OFFSET,
    .ccm_cdcrr_offset = MXC_CCM_CDCRR_OFFSET,
    .ccm_cacrr_offset = MXC_CCM_CACRR_OFFSET,
    .ccm_cdhipr_offset = MXC_CCM_CDHIPR_OFFSET,
    .prediv_mask = 0x1F800,
    .prediv_offset = 11,
    .prediv_val = 3,
    .div3ck_mask = 0xE0000000,
    .div3ck_offset = 29,
    .div3ck_val = 2,
    .emac_val = 0x08,
    .upthr_val = 25,
    .dnthr_val = 9,
    .pncthr_val = 33,
    .upcnt_val = 10,
    .dncnt_val = 10,
    .delay_time = 30,
};
```

`dvfs_core_data` is then passed as a parameter to `platform_device_register` and retrieved from the DVFSC driver using the Linux driver model semantics.

- For the ARD board, SW1 is declared in *arch/arm/mach-mx5/mx53_ard_pmic_ltc3589.c*:
-

```
/* CPU */
static struct regulator_consumer_supply sw1_consumers[] = {
    {
        .supply = "cpu_vcc",
    }
};

struct ltc3589;

static struct regulator_init_data sw1_init = {
    .constraints = {
        .name = "SW1",
        .min uV = 564000,
    }
};
```

```

        .max_uV = 1167000,
        .valid_ops_mask = REGULATOR_CHANGE_VOLTAGE,
        .valid_modes_mask = 0,
        .always_on = 1,
        .boot_on = 1,
        .initial_state = PM_SUSPEND_MEM,
        .state_mem = {
            .uV = 950000,
            .mode = REGULATOR_MODE_NORMAL,
            .enabled = 1,
        },
    },
    .num_consumer_supplies = ARRAY_SIZE(sw1_consumers),
    .consumer_supplies = sw1_consumers,
};

```

The most notable members of the structure are `min_uV` and `max_uV`. They define the voltage range allowed by the LTC3589 PMIC. When the CPU working point (WP) is changed, the voltage defined by the WP (`cpu_voltage` in the `cpu_wp_aec` structure) is checked against these values and set by the PMIC regulator driver.

To interface the PMIC with the DVFSC driver, the PMIC driver only has to register its regulators, defining the proper voltage ranges. The DVFSC driver interacts with PMIC drivers through the generic regulator API.

Software Interface

- To Enable the DVFS core use this command:
`echo 1 > /sys/devices/platform/mxc_dvfs_core.0/enable`
- To Disable The DVFS core use this command:
`echo 0 > /sys/devices/platform/mxc_dvfs_core.0/enable`

CPU Frequency Scaling (CPUFREQ) Driver

Most of the information in this paragraph has been copied from the Linux Reference Manual - Chapter 18. Please refer to this document for more information.

The CPU frequency scaling device driver allows the clock speed of the CPU to be changed on the fly. This driver functions in a very similar fashion as the DVFS core driver, except that the monitoring is done in software, through the *cpufreq* framework.

More information about cpufreq is available in the kernel documentation: <http://lxr.linux.no/#linux+v2.6.35.9/Documentation/cpu-freq/>

The most important files for the ARD platform are *governors.txt* and *user-guide.txt*.

Software Operation

The CPUFREQ driver dynamically changes the CPU frequency and voltage according to the working points (frequencies/voltages) defined in *arch/arm/mach-mx5/mx53_wp.c*. Please refer to the previous section for more information about these working points.

The driver is implemented in *arch/arm/plat-mxc/cpufreq.c*

The working points logic and interactions with the regulator framework are implemented in *set_cpu_freq()*, in a way that is very similar to the DVFS driver. It is therefore independent from the specifics of the PMIC (since the handle to the regulator supplying the CPU voltage is acquired in a generic way).

Kernel configuration

CONFIG_CPU_FREQ—In menuconfig, this option is located under *CPU Power Management > CPU Frequency scaling*.

The following options can be selected:

- CPU Frequency scaling
- CPU frequency translation statistics
- Default CPU frequency governor (userspace)
- Performance governor
- Powersave governor
- Userspace governor for userspace frequency scaling
- Conservative CPU frequency governor
- CPU frequency driver for i.MX CPUs

Software Interface

The behavior of the CPUFREQ driver can be changed using *sysfs*. The frequency and voltages are dynamically adjusted according to *governors*, which are software algorithms that control the working point according to given criteria (more information in the Linux documentation: <http://lxr.linux.no/#linux+v2.6.35.9/Documentation/cpu-freq/governors.txt>).

- To view what values the CPU frequency can be changed to in KHz (The values in the first column are the frequency values) use this command:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state
```
- To change the CPU frequency to a value that is given by using the command above (e.g. to 800 MHz) use this command:

```
echo 800000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

The frequency *800000* is in KHz.
- The maximum frequency can be checked using this command:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```
- Use the following command to view the current CPU frequency in KHz:


```
cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq
```

- Use the following command to view available governors:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/  
scaling_available_governors
```

- Use the following command to set governors:

```
echo conservative > /sys/devices/system/cpu/cpu0/cpufreq/  
scaling_governor
```

Note: The ARD board only supports one working point (800MHz), so the governors will not change the frequency. This feature has been successfully tested on the Quick Start Board, which uses the 1GHz version of the i.MX53 CPU.

Software Based Peripheral Domain Frequency Scaling

Most of the information in this paragraph has been copied from the Linux Reference Manual - Chapter 19. Please refer to this document for more information.

The frequency of the clocks in the peripheral domain can be changed using the software based Bus Frequency Scaling driver. Enabling this driver can significantly lower the power numbers in the LP domain. Depending on the platform, the voltage of the peripheral domain can also be dropped using the on board PMIC.

The SW will automatically lower the frequency of the various clocks in the peripheral domain based on which drivers are active (it is assumed that the drivers will use the clock API to enable/disable their clocks)

Software Interface

- To enable the SW based Bus Frequency Scaling (not needed to enter LPAPM mode) use this command:

```
echo 1 > /sys/devices/platform/busfreq.0/enable
```

- To disable the SW based Bus Frequency Scaling use this command:

```
echo 0 > /sys/devices/platform/busfreq.0/enable
```

Kernel configuration

There is no option for the SW based Bus Frequency Scaling driver, it included by default.

Software Operation

The driver is implemented in *arch/arm/mach-mx5/bus_freq.c*

This driver is very similar to the DVFSC driver with respect to the interactions with the PMIC. On the ARD board, it controls SW2 through the regulator API. This is defined in the *bus_freq_data* from *arch/arm/mach-mx5/mx53_ard.c*:

```
static struct mxc_bus_freq_platform_data bus_freq_data = {  
    .gp_reg_id = "SW1",  
    .lp_reg_id = "SW2",  
};
```

This structure is passed to the bus_freq driver using the Linux driver model semantics and *platform_device_register()*.

Study of the LTC3589 driver

The LTC3589 driver code is made of two main parts:

- The actual driver, which is board-agnostic (in *drivers/*)
- The platform glue, that ties the PMIC driver to the platform (in *arch/arm/mach-xxx*)

Since the latter is described in detail in *Porting LTC3589 to a mx5x-family board*, we will only focus on the actual driver.

The driver has been extracted from the Freescale kernel for the i.MX53 ARD board since it is not part of the mainline Linux kernel.

Kernel configuration

To enable the LTC3589 driver, select the following items in a *menuconfig*:

- *Multifunction device drivers > Support for Linear LTC3589 with I2C*
- *Voltage and Current Regulator Support > LTC3589 Regulator Support*

The corresponding configuration parameters are *CONFIG_REGULATOR_LTC3589*, *CONFIG_MFD_LTC3589_I2C* and *CONFIG_MFD_LTC3589*.

The driver is registered in the kernel build/configuration system using:

- *drivers/mfd/Makefile*
- *drivers/mfd/Kconfig*
- *drivers/regulator/Makefile*
- *drivers/regulator/Kconfig*

Source files

The driver is split into two parts:

- *drivers/mfd/ltc3589-i2c*:
 - Implements the communication with the PMIC using the I2C API. Since this API is generic, the driver does not need to know about the details of the I2C implementation for a given board.
 - This file typically does not need to be modified
 - The I2C address is a board specific setting and is defined by `ltc3589_i2c_device` in *arch/arm/mach-mx5/mx53_ard_pmic_ltc3589.c* for the ARD board.
- *drivers/regulator/ltc3589-regulator*:
 - This is where the regulator logic is implemented.
 - The regulators are registered in the board file (*arch/arm/mach-mx5/mx53_ard_pmic_ltc3589.c* for the ARD board) using `arch/arm/mach-mx5/mx53_ard_pmic_ltc3589.c()`. This function creates a platform device for each regulator. Since it gets matched with the platform driver registered in

ltc3589_regulator_driver, ltc3589_regulator_probe() is called and then registers it using the regulator API function regulator_register().

- All the regulators implemented by the LTC3589 driver are defined in the following structure:
-

```
static struct regulator_desc ltc3589_reg[] = {
    {
        .name = "SW1",
        .id = LTC3589_SW1,
        .ops = &lttc3589_sw_ops,
        .type = REGULATOR_VOLTAGE,
        .n_voltages = 0x1F + 1,
        .owner = THIS_MODULE,
    },
    {
        .name = "SW2",
        .id = LTC3589_SW2,
        .ops = &lttc3589_sw_ops,
        .type = REGULATOR_VOLTAGE,
        .n_voltages = 0x1F + 1,
        .owner = THIS_MODULE,
    },
    {
        .name = "SW3",
        .id = LTC3589_SW3,
        .ops = &lttc3589_sw_ops,
        .type = REGULATOR_VOLTAGE,
        .n_voltages = 0x1F + 1,
        .owner = THIS_MODULE,
    },
    {
        .name = "SW4",
        .id = LTC3589_SW4,
        .ops = &lttc3589_sw4_ops,
        .type = REGULATOR_VOLTAGE,
        .owner = THIS_MODULE,
    },
    {
        .name = "LD01_STBY",
        .id = LTC3589_LD01,
        .ops = &lttc3589_ldo13_ops,
        .type = REGULATOR_VOLTAGE,
        .owner = THIS_MODULE,
    },
},
```

```

{
    .name = "LDO2",
    .id = LTC3589_LDO2,
    .ops = &ltc3589_ldo2_ops,
    .type = REGULATOR_VOLTAGE,
    .n_voltages = 0x1F + 1,
    .owner = THIS_MODULE,
},
{
    .name = "LDO3",
    .id = LTC3589_LDO3,
    .ops = &ltc3589_ldo13_ops,
    .type = REGULATOR_VOLTAGE,
    .owner = THIS_MODULE,
},
{
    .name = "LDO4",
    .id = LTC3589_LDO4,
    .ops = &ltc3589_ldo4_ops,
    .type = REGULATOR_VOLTAGE,
    .n_voltages = ARRAY_SIZE(LDO4_VSEL_table),
    .owner = THIS_MODULE,
},
};

```

- Using the `.ops` field, a callback function is assigned for each regulator operation (e.g. get/set voltage/suspend...)
- **Note:** There is still one board-specific reference to `cpu_is_mx53_rev()` that should be removed when porting the code to other platforms. The values of the resistors hard-coded at the top of the file (`LTC3589_LDO2_Rx`) also need to be changed.

Using the driver

- When porting the driver to a different platform:
 - The files in *drivers* are board-agnostic and typically do not need to be modified. When using this driver on a new board, one typically wants to call `ltc3589_register_regulator()` in the board files (under *arch/arm*). This function is declared in *include/linux/mfd/ltc3589/core.h*.
- When using the regulator from a driver or from the platform code:
 - Regulators can be acquired and released using the regulator API (<http://lxr.linux.no/#linux+v2.6.35.9/Documentation/power/regulator/>): `regulator_get()` and `regulator_put()`. The voltages can be set using `regulator_set_voltage()`. Other settings can also be changed on runtime (please refer to the kernel documentation). The name of the regulator has to be

passed as a parameter and is usually put in platform data structures.

- A good example is the DVFS Core driver.