

System Controller Firmware API Reference Guide

i.MX8 QM/QP (A0)

Generated by Doxygen 1.8.15

Mon Nov 19 2018 13:10:55

1 Overview	1
1.1 System Initialization and Boot	2
1.2 System Controller Communication	2
1.3 System Controller Services	2
1.3.1 Power Management Service	2
1.3.2 Resource Management Service	3
1.3.3 Pad Configuration Service	3
1.3.4 Timer Service	4
1.3.5 Interrupt Service	4
1.3.6 Miscellaneous Service	4
2 Disclaimer	5
3 Usage	7
3.1 SCFW API	7
3.2 Loading	7
3.3 Boot Flags	7
4 Resource List	9
5 Clock List	21
6 Control List	27
7 Pad List	31
8 Module Index	41
8.1 Modules	41
9 File Index	43
9.1 File List	43
10 Module Documentation	45
10.1 (SVC) Interrupt Service	45
10.1.1 Detailed Description	47
10.1.2 Function Documentation	47
10.1.2.1 sc_irq_enable()	48
10.1.2.2 sc_irq_status()	48
10.2 (SVC) Miscellaneous Service	50
10.2.1 Detailed Description	53
10.2.2 Function Documentation	53
10.2.2.1 sc_misc_set_control()	53
10.2.2.2 sc_misc_get_control()	54

10.2.2.3 sc_misc_set_max_dma_group()	55
10.2.2.4 sc_misc_set_dma_group()	55
10.2.2.5 sc_misc_seco_image_load()	56
10.2.2.6 sc_misc_seco_authenticate()	57
10.2.2.7 sc_misc_seco_fuse_write()	57
10.2.2.8 sc_misc_seco_enable_debug()	58
10.2.2.9 sc_misc_seco_forward_lifecycle()	58
10.2.2.10 sc_misc_seco_return_lifecycle()	59
10.2.2.11 sc_misc_seco_build_info()	60
10.2.2.12 sc_misc_seco_chip_info()	60
10.2.2.13 sc_misc_seco_attest_mode()	60
10.2.2.14 sc_misc_seco_attest()	61
10.2.2.15 sc_misc_seco_get_attest_pkey()	62
10.2.2.16 sc_misc_seco_get_attest_sign()	62
10.2.2.17 sc_misc_seco_attest_verify()	63
10.2.2.18 sc_misc_seco_commit()	64
10.2.2.19 sc_misc_debug_out()	64
10.2.2.20 sc_misc_waveform_capture()	64
10.2.2.21 sc_misc_build_info()	65
10.2.2.22 sc_misc_unique_id()	65
10.2.2.23 sc_misc_set_ari()	66
10.2.2.24 sc_misc_boot_status()	66
10.2.2.25 sc_misc_boot_done()	67
10.2.2.26 sc_misc_otp_fuse_read()	67
10.2.2.27 sc_misc_otp_fuse_write()	68
10.2.2.28 sc_misc_set_temp()	69
10.2.2.29 sc_misc_get_temp()	69
10.2.2.30 sc_misc_get_boot_dev()	70
10.2.2.31 sc_misc_get_boot_type()	70
10.2.2.32 sc_misc_get_button_status()	71
10.2.2.33 sc_misc_rompatch_checksum()	71
10.3 (SVC) Pad Service	72
10.3.1 Detailed Description	75
10.3.2 Typedef Documentation	77
10.3.2.1 sc_pad_config_t	77
10.3.2.2 sc_pad_iso_t	77
10.3.2.3 sc_pad_28fdsoi_dse_t	77
10.3.2.4 sc_pad_28fdsoi_ps_t	77
10.3.2.5 sc_pad_28fdsoi_pus_t	77

10.3.3 Function Documentation	78
10.3.3.1 sc_pad_set_mux()	78
10.3.3.2 sc_pad_get_mux()	78
10.3.3.3 sc_pad_set_gp()	79
10.3.3.4 sc_pad_get_gp()	80
10.3.3.5 sc_pad_set_wakeup()	80
10.3.3.6 sc_pad_get_wakeup()	81
10.3.3.7 sc_pad_set_all()	81
10.3.3.8 sc_pad_get_all()	82
10.3.3.9 sc_pad_set()	83
10.3.3.10 sc_pad_get()	84
10.3.3.11 sc_pad_set_gp_28fdsoi()	84
10.3.3.12 sc_pad_get_gp_28fdsoi()	85
10.3.3.13 sc_pad_set_gp_28fdsoi_hsic()	86
10.3.3.14 sc_pad_get_gp_28fdsoi_hsic()	86
10.3.3.15 sc_pad_set_gp_28fdsoi_comp()	87
10.3.3.16 sc_pad_get_gp_28fdsoi_comp()	88
10.4 (SVC) Power Management Service	90
10.4.1 Detailed Description	95
10.4.2 Macro Definition Documentation	96
10.4.2.1 SC_PM_CLK_MODE_ROM_INIT	96
10.4.2.2 SC_PM_CLK_MODE_ON	96
10.4.2.3 SC_PM_PARENT_XTAL	97
10.4.2.4 SC_PM_PARENT_BYPS	97
10.4.3 Typedef Documentation	97
10.4.3.1 sc_pm_power_mode_t	97
10.4.4 Function Documentation	97
10.4.4.1 sc_pm_set_sys_power_mode()	97
10.4.4.2 sc_pm_set_partition_power_mode()	98
10.4.4.3 sc_pm_get_sys_power_mode()	99
10.4.4.4 sc_pm_set_resource_power_mode()	99
10.4.4.5 sc_pm_set_resource_power_mode_all()	100
10.4.4.6 sc_pm_get_resource_power_mode()	101
10.4.4.7 sc_pm_req_low_power_mode()	101
10.4.4.8 sc_pm_req_cpu_low_power_mode()	102
10.4.4.9 sc_pm_set_cpu_resume_addr()	102
10.4.4.10 sc_pm_set_cpu_resume()	103
10.4.4.11 sc_pm_req_sys_if_power_mode()	103
10.4.4.12 sc_pm_set_clock_rate()	104

10.4.4.13	sc_pm_get_clock_rate()	105
10.4.4.14	sc_pm_clock_enable()	105
10.4.4.15	sc_pm_set_clock_parent()	106
10.4.4.16	sc_pm_get_clock_parent()	107
10.4.4.17	sc_pm_reset()	108
10.4.4.18	sc_pm_reset_reason()	108
10.4.4.19	sc_pm_boot()	109
10.4.4.20	sc_pm_reboot()	109
10.4.4.21	sc_pm_reboot_partition()	110
10.4.4.22	sc_pm_cpu_start()	111
10.5	(SVC) Resource Management Service	112
10.5.1	Detailed Description	115
10.5.2	Typedef Documentation	117
10.5.2.1	sc_rm_perm_t	118
10.5.3	Function Documentation	118
10.5.3.1	sc_rm_partition_alloc()	118
10.5.3.2	sc_rm_set_confidential()	119
10.5.3.3	sc_rm_partition_free()	119
10.5.3.4	sc_rm_get_did()	120
10.5.3.5	sc_rm_partition_static()	120
10.5.3.6	sc_rm_partition_lock()	122
10.5.3.7	sc_rm_get_partition()	123
10.5.3.8	sc_rm_set_parent()	123
10.5.3.9	sc_rm_move_all()	124
10.5.3.10	sc_rm_assign_resource()	124
10.5.3.11	sc_rm_set_resource_movable()	125
10.5.3.12	sc_rm_set_subsys_rsrc_movable()	126
10.5.3.13	sc_rm_set_master_attributes()	126
10.5.3.14	sc_rm_set_master_sid()	127
10.5.3.15	sc_rm_set_peripheral_permissions()	128
10.5.3.16	sc_rm_is_resource_owned()	128
10.5.3.17	sc_rm_is_resource_master()	129
10.5.3.18	sc_rm_is_resource_peripheral()	129
10.5.3.19	sc_rm_get_resource_info()	130
10.5.3.20	sc_rm_memreg_alloc()	130
10.5.3.21	sc_rm_memreg_split()	131
10.5.3.22	sc_rm_memreg_free()	132
10.5.3.23	sc_rm_find_memreg()	132
10.5.3.24	sc_rm_assign_memreg()	133

10.5.3.25 sc_rm_set_memreg_permissions()	134
10.5.3.26 sc_rm_is_memreg_owned()	134
10.5.3.27 sc_rm_get_memreg_info()	135
10.5.3.28 sc_rm_assign_pad()	135
10.5.3.29 sc_rm_set_pad_movable()	136
10.5.3.30 sc_rm_is_pad_owned()	137
10.5.3.31 sc_rm_dump()	137
10.6 (SVC) Timer Service	138
10.6.1 Detailed Description	139
10.6.2 Function Documentation	139
10.6.2.1 sc_timer_set_wdog_timeout()	139
10.6.2.2 sc_timer_set_wdog_pre_timeout()	140
10.6.2.3 sc_timer_start_wdog()	140
10.6.2.4 sc_timer_stop_wdog()	141
10.6.2.5 sc_timer_ping_wdog()	141
10.6.2.6 sc_timer_get_wdog_status()	142
10.6.2.7 sc_timer_pt_get_wdog_status()	142
10.6.2.8 sc_timer_set_wdog_action()	143
10.6.2.9 sc_timer_set_rtc_time()	143
10.6.2.10 sc_timer_get_rtc_time()	144
10.6.2.11 sc_timer_get_rtc_sec1970()	145
10.6.2.12 sc_timer_set_rtc_alarm()	145
10.6.2.13 sc_timer_set_rtc_periodic_alarm()	146
10.6.2.14 sc_timer_cancel_rtc_alarm()	146
10.6.2.15 sc_timer_set_rtc_calb()	147
10.6.2.16 sc_timer_set_sysctr_alarm()	147
10.6.2.17 sc_timer_set_sysctr_periodic_alarm()	148
10.6.2.18 sc_timer_cancel_sysctr_alarm()	148
11 File Documentation	151
11.1 platform/main/ipc.h File Reference	151
11.1.1 Detailed Description	151
11.1.2 Function Documentation	151
11.1.2.1 sc_ipc_open()	151
11.1.2.2 sc_ipc_close()	152
11.1.2.3 sc_ipc_read()	152
11.1.2.4 sc_ipc_write()	152
11.2 platform/main/types.h File Reference	154
11.2.1 Detailed Description	170

11.2.2 Typedef Documentation	170
11.2.2.1 sc_rsrc_t	170
11.2.2.2 sc_pad_t	170
11.3 platform/svc/irq/api.h File Reference	170
11.3.1 Detailed Description	173
11.4 platform/svc/misc/api.h File Reference	173
11.4.1 Detailed Description	176
11.5 platform/svc/pad/api.h File Reference	176
11.5.1 Detailed Description	179
11.6 platform/svc/pm/api.h File Reference	179
11.6.1 Detailed Description	184
11.7 platform/svc/rm/api.h File Reference	184
11.7.1 Detailed Description	187
11.8 platform/svc/timer/api.h File Reference	187
11.8.1 Detailed Description	189
Index	191

Chapter 1

Overview

The System Controller (SC) provides an abstraction to many of the underlying features of the hardware. This function runs on a Cortex-M processor which executes SC firmware (FW). This overview describes the features of the SCFW and the APIs exposed to other software components.

Features include:

- [System Initialization and Boot](#)
- [System Controller Communication](#)
- [Power Management](#)
- [Resource Management](#)
- [Pad Configuration](#)
- [Timers](#)
- [Interrupts](#)
- [Miscellaneous](#)

Due to this abstraction, some HW described in the SoC RM that is used by the SCFW is not directly accessible to other cores. This includes:

- All resources in the SCU subsystem (SCU M4, SCU LPUART, SCU LPI2C, etc.).
- All resource accessed via MSI links from the SCU subsystem (inc. pads, DSC, XRDC2, eCSR)
- OCRM controller, CAAM MP, eDMA MP & LPCG
- DB STC & LPCG, IMG GPR
- GIC/IRQSTR LPCG, IRQSTR.SCU and IRQSTR.CTI
- Any other resources reserved by the port of the SCFW to the board

1.1 System Initialization and Boot

The SC firmware runs on the SCU immediately after the SCU Read-only-memory (ROM) finishes loading code/data images from the first container. It is responsible for initializing many aspects of the system. This includes additional power and clock configuration and resource isolation hardware configuration. By default, the SC firmware configures the primary boot core to own most of the resources and launches the boot core. Additional configuration can be done by boot code.

1.2 System Controller Communication

Other software components in the system communicate to the SC via an exposed API library. This library is implemented to make Remote Procedure Calls (RPC) via an underlying Inter-Processor Communication (IPC) mechanism. The IPC is facilitated by a hardware-based mailbox system.

Software components (Linux, QNX, FreeRTOS, KSDK) delivered for i.MX8 already include ports of the client API. Other 3rd parties will need to first port the API to their environment before the API can be used. The porting kit release includes archives of the client API for existing SW. These can be used as reference for porting the client API. All that needs to be implemented is the IPC layer which will utilize messaging units (MU) to communicate with the SCFW.

1.3 System Controller Services

The SCFW provides API access to all the system controller functionality exported to other software systems. This includes:

1.3.1 Power Management Service

All aspects of power management including power control, bias control, clock control, reset control, and wake-up event monitoring are grouped within the SC Power Management service.

- **Power Control** - The SC firmware is responsible for centralized management of power controls and external power management devices. It manages the power state and voltage of power domains as well as bias control. It also resets peripherals as required due to power state transitions. This is all done via the API by communicating power state needs for individual resources.
- **Clock Control** - The SC firmware is responsible for centralized management of clock controls. This includes clock sources such as oscillators and PLLs as well as clock dividers, muxes, and gates. This is all done via the API by communicating clocking needs for individual resources.
- **Reset Control** - The SC firmware is responsible for reset control. This includes booting/rebooting a partition, obtaining reset reasons, and starting/stopping of CPUs.

Before any hardware in the SoC can be used, SW must first power up the resource and enable any clocks that it requires, otherwise access will generate a bus error. The Power Management (PM) API is documented [here](#).

1.3.2 Resource Management Service

SC firmware is responsible for managing ownership and access permissions to system resources. The features of the resource management service supported by SC firmware include:

- Management of system resources such as SoC peripherals, memory regions, and pads
- Allows resources to be partitioned into different ownership groupings that are associated with different execution environments including multiple operating systems executing on different cores, TrustZone, and hypervisor
- Associates ownership with requests from messaging units within a resource partition
- Allows memory to be divided into memory regions that are then managed like other resources
- Allows owners to configure access permissions to resources
- Configures hardware components to provide hardware enforced isolation
- Configures hardware components to directly control secure/nonsecure attribute driven on bus fabric
- Provides ownership and access permission information to other system controller functions (e.g. pad ownership information to the pad muxing functions)

Protection of resources is provided in two ways. First, the SCFW itself checks resource access rights when API calls are made that affect a specific resource. Depending on the API call, this may require that the caller be the owner, parent of the owner, or an ancestor of the owner. Second, any hardware available to enforce access controls is configured based on the RM state. This includes the configuration of IP such as XRDC2, XRDC, or RDC, as well as management pages of IP like CAAM.

The Resource Management (RM) API is documented [here](#).

1.3.3 Pad Configuration Service

Pad configuration is managed by SC firmware. The pad configuration features supported by the SC firmware include:

- Configuring the mux, input/output connection, and low-power isolation mode.
- Configuring the technology-specific pad setting such as drive strength, pullup/pulldown, etc.
- Configuring compensation for pad groups with dual voltage capability.

The Pad (PAD) API is documented [here](#).

1.3.4 Timer Service

Many timer oriented services are grouped within the SC Timer service. This includes watchdogs, RTC, and system counter.

- **Watchdog** - The SC firmware provides "virtual" watchdogs for all execution environments. Features include update of the watchdog timeout, start/stop of the watchdog, refresh of the watchdog, return of the watchdog status such as maximum watchdog timeout that can be set, watchdog timeout interval, and watchdog timeout interval remaining.
- **Real-Time-Clock** - The SC firmware is responsible for providing access to the RTC. Features include setting the time, getting the time, and setting alarms.
- **System Counter** - The SC firmware is responsible for providing access to the SYSCTR. Features include setting an absolute alarm or a relative, periodic alarm. Reading is done directly via local hardware interfaces available for each CPU.

The Timer API is documented [here](#).

1.3.5 Interrupt Service

The System Controller needs a method to inform users about asynchronous notification events. This is done via the Interrupt service. The service provides APIs to enable/disable interrupts to the user and to read the status of pending interrupts. Reading the status automatically clears any pending state. The Interrupt (IRQ) API is documented [here](#).

1.3.6 Miscellaneous Service

On previous i.MX devices, miscellaneous features were controlled using IOMUX GPR registers with signals connected to configurable hardware. This functionality is being replaced with DSC GPR signals. SC firmware is responsible for programming the GPR signals to configure these subsystem features. The SC firmware also responsible for monitoring various temperature, voltage, and clock sensors.

- **Controls** - The SC firmware provides access to miscellaneous controls. Features include software request to set (write) miscellaneous controls and software request to get (read) miscellaneous controls.
- **Security** - The SC firmware provides access to several security functions including image loading and authentication.
- **DMA** - The SC firmware provides access to DMA channel grouping and priority functions.
- **Temp** - The SC firmware provides access to temperature sensors.

The Miscellaneous (MISC) API is documented [here](#).

Chapter 2

Disclaimer

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein. NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions. While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREE↵NCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE P↵LUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C 5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobile↵GT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, AMBA, Arm Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and ?Vision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. Arm7, Arm9, Arm11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, Mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

(c) 2018 NXP B.V.



Chapter 3

Usage

3.1 SCFW API

Calling the functions in the SCFW requires a client API which utilizes an RPC/IPC layer to communicate to the SCFW binary running on the SCU. Application environments (Linux, QNX, FreeRTOS, KSDK) delivered for i.MX8 already include ports of the client API. Other 3rd parties will need to first port the API to their environment before the API can be used. The porting kit release includes archives of the client API for existing SW. These can be used as reference for porting the client API. All that needs to be implemented is the IPC layer which will utilize messaging units (MU) to communicate with the SCFW.

The SCFW API is documented in the individual API sections. There are examples in the Details section for each service.

- [Resource Management](#)
- [Power Management](#)
- [Pad Configuration](#)
- [Timers](#)
- [Interrupts](#)
- [Miscellaneous](#)

3.2 Loading

The SCFW is loaded by including the binary (scfw_tcm.bin) in the boot container.

3.3 Boot Flags

The image container holding the SCFW should also have the boot flags configured. This is a set of flags (32-bits) specified with the -flags option to mkimage.

These are defined as:

Flag	Bit	Meaning
SC_BD_FLAGS_NOT_SECURE	16	Initial boot partition is not secure
SC_BD_FLAGS_NOT_ISOLATED	17	Initial boot partition is not isolated
SC_BD_FLAGS_RESTRICTED	18	Initial boot partition is restricted
SC_BD_FLAGS_GRANT	19	Initial boot partition grants access to the SCFW
SC_BD_FLAGS_NOT_COHERENT	20	Initial boot partition is not coherent
SC_BD_FLAGS_ALT_CONFIG	21	Alternate SCFW config (passed to board.c)
SC_BD_FLAGS_EARLY_CPU_START	22	Start some CPUs early
SC_BD_FLAGS_DDRTEST	23	Config for DDR stress test
SC_BD_FLAGS_NO_AP	24	Don't boot AP even if requested by ROM

See the [sc_rm_partition_alloc\(\)](#) function for more info. Note some of the flags are inverted before calling this function! This results in a default (all 0) which is appropriate for normal OS execution. That is a partition which is secure, isolated, not restricted, not grant, coherent, no early start, and no DDR test.

Chapter 4

Resource List

The table below lists the resources available in the system (SoC specific). Resource is a parameter to many API calls.

See also

[\(SVC\) Resource Management Service](#)

[\(SVC\) Power Management Service](#)

Resource	Subsystem	Instance	Description
SC_R_A53	A53	0	Cluster
SC_R_A53_0	A53	0	CPU 0
SC_R_A53_1	A53	0	CPU 1
SC_R_A53_2	A53	0	CPU 2
SC_R_A53_3	A53	0	CPU 3
SC_R_A72	A72	0	Cluster
SC_R_A72_0	A72	0	CPU 0
SC_R_A72_1	A72	0	CPU 1
SC_R_ADC_0	DMA	0	ADC 0
SC_R_ADC_1	DMA	0	ADC 1
SC_R_AMIX	AUDIO	0	Audio mixer
SC_R_ASRC_0	AUDIO	0	ASRC 0
SC_R_ASRC_1	AUDIO	0	ASRC 1
SC_R_ATTESTATION	SC	0	Attestation
SC_R_AUDIO_CLK_0	AUDIO	0	Audio clock 0
SC_R_AUDIO_CLK_1	AUDIO	0	Audio clock 1
SC_R_AUDIO_PLL_0	AUDIO	0	Audio PLL 0
SC_R_AUDIO_PLL_1	AUDIO	0	Audio PLL 1
SC_R_BOARD_R0	BOARD	0	Misc. board component 0
SC_R_BOARD_R1	BOARD	0	Misc. board component 1
SC_R_BOARD_R2	BOARD	0	Misc. board component 2
SC_R_BOARD_R3	BOARD	0	Misc. board component 3
SC_R_BOARD_R4	BOARD	0	Misc. board component 4
SC_R_BOARD_R5	BOARD	0	Misc. board component 5

Resource	Subsystem	Instance	Description
SC_R_BOARD_R6	BOARD	0	Misc. board component 6
SC_R_BOARD_R7	BOARD	0	Misc. board component 7
SC_R_CAAM_JR0	SC	0	CAAM job ring 0
SC_R_CAAM_JR0_OUT	SC	0	CAAM job ring 0 out
SC_R_CAAM_JR1	SC	0	CAAM job ring 1
SC_R_CAAM_JR1_OUT	SC	0	CAAM job ring 1 out
SC_R_CAAM_JR2	SC	0	CAAM job ring 2
SC_R_CAAM_JR2_OUT	SC	0	CAAM job ring 2 out
SC_R_CAAM_JR3	SC	0	CAAM job ring 3
SC_R_CAAM_JR3_OUT	SC	0	CAAM job ring 3 out
SC_R_CAN_0	DMA	0	CAN 0
SC_R_CAN_1	DMA	0	CAN 1
SC_R_CAN_2	DMA	0	CAN 2
SC_R_CCI	CCI	0	CCI
SC_R_CSI_0	CSI	0	CSI
SC_R_CSI_0_I2C_0	CSI	0	I2C
SC_R_CSI_0_PWM_0	CSI	0	PWM
SC_R_CSI_1	CSI	1	CSI
SC_R_CSI_1_I2C_0	CSI	1	I2C
SC_R_CSI_1_PWM_0	CSI	1	PWM
SC_R_DB	DB	0	DB
SC_R_DBLOGIC	DBLOGIC	0	DBLOGIC
SC_R_DC_0	DC	0	DC
SC_R_DC_0_BLIT0	DC	0	BLIT0
SC_R_DC_0_BLIT1	DC	0	BLIT1
SC_R_DC_0_BLIT2	DC	0	BLIT2
SC_R_DC_0_BLIT_OUT	DC	0	BLIT OUT
SC_R_DC_0_FRAC0	DC	0	FRAC0
SC_R_DC_0_PLL_0	DC	0	PLL 0
SC_R_DC_0_PLL_1	DC	0	PLL 1
SC_R_DC_0_VIDEO0	DC	0	VIDEO0
SC_R_DC_0_VIDEO1	DC	0	VIDEO1
SC_R_DC_0_WARP	DC	0	WARP
SC_R_DC_1	DC	1	DC
SC_R_DC_1_BLIT0	DC	1	BLIT0
SC_R_DC_1_BLIT1	DC	1	BLIT1
SC_R_DC_1_BLIT2	DC	1	BLIT2
SC_R_DC_1_BLIT_OUT	DC	1	BLIT OUT
SC_R_DC_1_FRAC0	DC	1	FRAC0
SC_R_DC_1_PLL_0	DC	1	PLL 0
SC_R_DC_1_PLL_1	DC	1	PLL 1
SC_R_DC_1_VIDEO0	DC	1	VIDEO0
SC_R_DC_1_VIDEO1	DC	1	VIDEO1
SC_R_DC_1_WARP	DC	1	WARP
SC_R_DEBUG	SC	0	Debug
SC_R_DMA_0_CH0	DMA	0	DMA 0 channel 0
SC_R_DMA_0_CH1	DMA	0	DMA 0 channel 1

Resource	Subsystem	Instance	Description
SC_R_DMA_0_CH10	DMA	0	DMA 0 channel 10
SC_R_DMA_0_CH11	DMA	0	DMA 0 channel 11
SC_R_DMA_0_CH12	DMA	0	DMA 0 channel 12
SC_R_DMA_0_CH13	DMA	0	DMA 0 channel 13
SC_R_DMA_0_CH14	DMA	0	DMA 0 channel 14
SC_R_DMA_0_CH15	DMA	0	DMA 0 channel 15
SC_R_DMA_0_CH16	DMA	0	DMA 0 channel 16
SC_R_DMA_0_CH17	DMA	0	DMA 0 channel 17
SC_R_DMA_0_CH18	DMA	0	DMA 0 channel 18
SC_R_DMA_0_CH19	DMA	0	DMA 0 channel 19
SC_R_DMA_0_CH2	DMA	0	DMA 0 channel 2
SC_R_DMA_0_CH20	DMA	0	DMA 0 channel 20
SC_R_DMA_0_CH21	DMA	0	DMA 0 channel 21
SC_R_DMA_0_CH22	DMA	0	DMA 0 channel 22
SC_R_DMA_0_CH23	DMA	0	DMA 0 channel 23
SC_R_DMA_0_CH24	DMA	0	DMA 0 channel 24
SC_R_DMA_0_CH25	DMA	0	DMA 0 channel 25
SC_R_DMA_0_CH26	DMA	0	DMA 0 channel 26
SC_R_DMA_0_CH27	DMA	0	DMA 0 channel 27
SC_R_DMA_0_CH28	DMA	0	DMA 0 channel 28
SC_R_DMA_0_CH29	DMA	0	DMA 0 channel 29
SC_R_DMA_0_CH3	DMA	0	DMA 0 channel 3
SC_R_DMA_0_CH30	DMA	0	DMA 0 channel 30
SC_R_DMA_0_CH31	DMA	0	DMA 0 channel 31
SC_R_DMA_0_CH4	DMA	0	DMA 0 channel 4
SC_R_DMA_0_CH5	DMA	0	DMA 0 channel 5
SC_R_DMA_0_CH6	DMA	0	DMA 0 channel 6
SC_R_DMA_0_CH7	DMA	0	DMA 0 channel 7
SC_R_DMA_0_CH8	DMA	0	DMA 0 channel 8
SC_R_DMA_0_CH9	DMA	0	DMA 0 channel 9
SC_R_DMA_1_CH0	DMA	0	DMA 1 channel 0
SC_R_DMA_1_CH1	DMA	0	DMA 1 channel 1
SC_R_DMA_1_CH10	DMA	0	DMA 1 channel 10
SC_R_DMA_1_CH11	DMA	0	DMA 1 channel 11
SC_R_DMA_1_CH12	DMA	0	DMA 1 channel 12
SC_R_DMA_1_CH13	DMA	0	DMA 1 channel 13
SC_R_DMA_1_CH14	DMA	0	DMA 1 channel 14
SC_R_DMA_1_CH15	DMA	0	DMA 1 channel 15
SC_R_DMA_1_CH16	DMA	0	DMA 1 channel 16
SC_R_DMA_1_CH17	DMA	0	DMA 1 channel 17
SC_R_DMA_1_CH18	DMA	0	DMA 1 channel 18
SC_R_DMA_1_CH19	DMA	0	DMA 1 channel 19
SC_R_DMA_1_CH2	DMA	0	DMA 1 channel 2
SC_R_DMA_1_CH20	DMA	0	DMA 1 channel 20
SC_R_DMA_1_CH21	DMA	0	DMA 1 channel 21
SC_R_DMA_1_CH22	DMA	0	DMA 1 channel 22
SC_R_DMA_1_CH23	DMA	0	DMA 1 channel 23

Resource	Subsystem	Instance	Description
SC_R_DMA_1_CH24	DMA	0	DMA 1 channel 24
SC_R_DMA_1_CH25	DMA	0	DMA 1 channel 25
SC_R_DMA_1_CH26	DMA	0	DMA 1 channel 26
SC_R_DMA_1_CH27	DMA	0	DMA 1 channel 27
SC_R_DMA_1_CH28	DMA	0	DMA 1 channel 28
SC_R_DMA_1_CH29	DMA	0	DMA 1 channel 29
SC_R_DMA_1_CH3	DMA	0	DMA 1 channel 3
SC_R_DMA_1_CH30	DMA	0	DMA 1 channel 30
SC_R_DMA_1_CH31	DMA	0	DMA 1 channel 31
SC_R_DMA_1_CH4	DMA	0	DMA 1 channel 4
SC_R_DMA_1_CH5	DMA	0	DMA 1 channel 5
SC_R_DMA_1_CH6	DMA	0	DMA 1 channel 6
SC_R_DMA_1_CH7	DMA	0	DMA 1 channel 7
SC_R_DMA_1_CH8	DMA	0	DMA 1 channel 8
SC_R_DMA_1_CH9	DMA	0	DMA 1 channel 9
SC_R_DMA_2_CH0	AUDIO	0	DMA 0 channel 0
SC_R_DMA_2_CH1	AUDIO	0	DMA 0 channel 1
SC_R_DMA_2_CH10	AUDIO	0	DMA 0 channel 10
SC_R_DMA_2_CH11	AUDIO	0	DMA 0 channel 11
SC_R_DMA_2_CH12	AUDIO	0	DMA 0 channel 12
SC_R_DMA_2_CH13	AUDIO	0	DMA 0 channel 13
SC_R_DMA_2_CH14	AUDIO	0	DMA 0 channel 14
SC_R_DMA_2_CH15	AUDIO	0	DMA 0 channel 15
SC_R_DMA_2_CH16	AUDIO	0	DMA 0 channel 16
SC_R_DMA_2_CH17	AUDIO	0	DMA 0 channel 17
SC_R_DMA_2_CH18	AUDIO	0	DMA 0 channel 18
SC_R_DMA_2_CH19	AUDIO	0	DMA 0 channel 19
SC_R_DMA_2_CH2	AUDIO	0	DMA 0 channel 2
SC_R_DMA_2_CH20	AUDIO	0	DMA 0 channel 20
SC_R_DMA_2_CH21	AUDIO	0	DMA 0 channel 21
SC_R_DMA_2_CH22	AUDIO	0	DMA 0 channel 22
SC_R_DMA_2_CH23	AUDIO	0	DMA 0 channel 23
SC_R_DMA_2_CH24	AUDIO	0	DMA 0 channel 24
SC_R_DMA_2_CH25	AUDIO	0	DMA 0 channel 25
SC_R_DMA_2_CH26	AUDIO	0	DMA 0 channel 26
SC_R_DMA_2_CH27	AUDIO	0	DMA 0 channel 27
SC_R_DMA_2_CH28	AUDIO	0	DMA 0 channel 28
SC_R_DMA_2_CH29	AUDIO	0	DMA 0 channel 29
SC_R_DMA_2_CH3	AUDIO	0	DMA 0 channel 3
SC_R_DMA_2_CH30	AUDIO	0	DMA 0 channel 30
SC_R_DMA_2_CH31	AUDIO	0	DMA 0 channel 31
SC_R_DMA_2_CH4	AUDIO	0	DMA 0 channel 4
SC_R_DMA_2_CH5	AUDIO	0	DMA 0 channel 5
SC_R_DMA_2_CH6	AUDIO	0	DMA 0 channel 6
SC_R_DMA_2_CH7	AUDIO	0	DMA 0 channel 7
SC_R_DMA_2_CH8	AUDIO	0	DMA 0 channel 8
SC_R_DMA_2_CH9	AUDIO	0	DMA 0 channel 9

Resource	Subsystem	Instance	Description
SC_R_DMA_3_CH0	AUDIO	0	DMA 1 channel 0
SC_R_DMA_3_CH1	AUDIO	0	DMA 1 channel 1
SC_R_DMA_3_CH10	AUDIO	0	DMA 1 channel 10
SC_R_DMA_3_CH11	AUDIO	0	DMA 1 channel 11
SC_R_DMA_3_CH12	AUDIO	0	DMA 1 channel 12
SC_R_DMA_3_CH13	AUDIO	0	DMA 1 channel 13
SC_R_DMA_3_CH14	AUDIO	0	DMA 1 channel 14
SC_R_DMA_3_CH15	AUDIO	0	DMA 1 channel 15
SC_R_DMA_3_CH16	AUDIO	0	DMA 1 channel 16
SC_R_DMA_3_CH17	AUDIO	0	DMA 1 channel 17
SC_R_DMA_3_CH18	AUDIO	0	DMA 1 channel 18
SC_R_DMA_3_CH19	AUDIO	0	DMA 1 channel 19
SC_R_DMA_3_CH2	AUDIO	0	DMA 1 channel 2
SC_R_DMA_3_CH20	AUDIO	0	DMA 1 channel 20
SC_R_DMA_3_CH21	AUDIO	0	DMA 1 channel 21
SC_R_DMA_3_CH22	AUDIO	0	DMA 1 channel 22
SC_R_DMA_3_CH23	AUDIO	0	DMA 1 channel 23
SC_R_DMA_3_CH24	AUDIO	0	DMA 1 channel 24
SC_R_DMA_3_CH25	AUDIO	0	DMA 1 channel 25
SC_R_DMA_3_CH26	AUDIO	0	DMA 1 channel 26
SC_R_DMA_3_CH27	AUDIO	0	DMA 1 channel 27
SC_R_DMA_3_CH28	AUDIO	0	DMA 1 channel 28
SC_R_DMA_3_CH29	AUDIO	0	DMA 1 channel 29
SC_R_DMA_3_CH3	AUDIO	0	DMA 1 channel 3
SC_R_DMA_3_CH30	AUDIO	0	DMA 1 channel 30
SC_R_DMA_3_CH31	AUDIO	0	DMA 1 channel 31
SC_R_DMA_3_CH4	AUDIO	0	DMA 1 channel 4
SC_R_DMA_3_CH5	AUDIO	0	DMA 1 channel 5
SC_R_DMA_3_CH6	AUDIO	0	DMA 1 channel 6
SC_R_DMA_3_CH7	AUDIO	0	DMA 1 channel 7
SC_R_DMA_3_CH8	AUDIO	0	DMA 1 channel 8
SC_R_DMA_3_CH9	AUDIO	0	DMA 1 channel 9
SC_R_DMA_4_CH0	CONN	0	DMA channel 0
SC_R_DMA_4_CH1	CONN	0	DMA channel 1
SC_R_DMA_4_CH2	CONN	0	DMA channel 2
SC_R_DMA_4_CH3	CONN	0	DMA channel 3
SC_R_DMA_4_CH4	CONN	0	DMA channel 4
SC_R_DRC_0	DRC	0	DDR controller/phy
SC_R_DRC_1	DRC	1	DDR controller/phy
SC_R_DTCP	CONN	0	DTCP
SC_R_EMVSIM_0	DMA	0	EMVSIM 0
SC_R_EMVSIM_1	DMA	0	EMVSIM 1
SC_R_ENET_0	CONN	0	ENET 1
SC_R_ENET_1	CONN	0	ENET 2
SC_R_ESAI_0	AUDIO	0	ESAI 0
SC_R_ESAI_1	AUDIO	0	ESAI 1
SC_R_FSPI_0	LSIO	0	Flexible Serial Peripheral Interface (FSPI) 0

Resource	Subsystem	Instance	Description
SC_R_FSPI_1	LSIO	0	FSPI 1
SC_R_FTM_0	DMA	0	FTM 0
SC_R_FTM_1	DMA	0	FTM 1
SC_R_GIC	DBLOGIC	0	GIC
SC_R_GIC_SMMU	DBLOGIC	0	GIC & SMMU
SC_R_GPIO_0	LSIO	0	General Purpose I/O (GPIO) 0
SC_R_GPIO_1	LSIO	0	GPIO 1
SC_R_GPIO_2	LSIO	0	GPIO 2
SC_R_GPIO_3	LSIO	0	GPIO 3
SC_R_GPIO_4	LSIO	0	GPIO 4
SC_R_GPIO_5	LSIO	0	GPIO 5
SC_R_GPIO_6	LSIO	0	GPIO 6
SC_R_GPIO_7	LSIO	0	GPIO 7
SC_R_GPT_0	LSIO	0	General Purpose Timer (GPT) 0
SC_R_GPT_1	LSIO	0	GPT 1
SC_R_GPT_10	AUDIO	0	GPT 5
SC_R_GPT_2	LSIO	0	GPT 2
SC_R_GPT_3	LSIO	0	GPT 3
SC_R_GPT_4	LSIO	0	GPT 4
SC_R_GPT_5	AUDIO	0	GPT 0
SC_R_GPT_6	AUDIO	0	GPT 1
SC_R_GPT_7	AUDIO	0	GPT 2
SC_R_GPT_8	AUDIO	0	GPT 3
SC_R_GPT_9	AUDIO	0	GPT 4
SC_R_GPU_0_PID0	GPU	0	PID0
SC_R_GPU_0_PID1	GPU	0	PID1
SC_R_GPU_0_PID2	GPU	0	PID2
SC_R_GPU_0_PID3	GPU	0	PID3
SC_R_GPU_1_PID0	GPU	1	PID0
SC_R_GPU_1_PID1	GPU	1	PID1
SC_R_GPU_1_PID2	GPU	1	PID2
SC_R_GPU_1_PID3	GPU	1	PID3
SC_R_HDMI	HDMI	0	HDMI
SC_R_HDMI_I2C_0	HDMI	0	I2C
SC_R_HDMI_I2S	HDMI	0	HDMI I2S
SC_R_HDMI_PLL_0	HDMI	0	PLL 0
SC_R_HDMI_PLL_1	HDMI	0	PLL 1
SC_R_HDMI_RX	HDMI_RX	0	HDMI_RX
SC_R_HDMI_RX_BYPASS	HDMI_RX	0	HDMI RX Bypass
SC_R_HDMI_RX_I2C_0	HDMI_RX	0	I2C
SC_R_HDMI_RX_PWM_0	HDMI_RX	0	PWM
SC_R_HSIO_GPIO	HSIO	0	GPIO/MISC
SC_R_I2C_0	DMA	0	I2C 0
SC_R_I2C_1	DMA	0	I2C 1
SC_R_I2C_2	DMA	0	I2C 2
SC_R_I2C_3	DMA	0	I2C 3
SC_R_I2C_4	DMA	0	I2C 4

Resource	Subsystem	Instance	Description
SC_R_IEE	LSIO	0	Inline Encryption Engine (IEE)
SC_R_IEE_R0	LSIO	0	IEE region 0
SC_R_IEE_R1	LSIO	0	IEE region 1
SC_R_IEE_R2	LSIO	0	IEE region 2
SC_R_IEE_R3	LSIO	0	IEE region 3
SC_R_IEE_R4	LSIO	0	IEE region 4
SC_R_IEE_R5	LSIO	0	IEE region 5
SC_R_IEE_R6	LSIO	0	IEE region 6
SC_R_IEE_R7	LSIO	0	IEE region 7
SC_R_IRQSTR_DSP	DBLOGIC	0	IRQSTR DSP
SC_R_IRQSTR_M4_0	DBLOGIC	0	IRQSTR M4 0
SC_R_IRQSTR_M4_1	DBLOGIC	0	IRQSTR M4 1
SC_R_IRQSTR_SCU2	DBLOGIC	0	IRQSTR SCU2/WAKE
SC_R_ISI_CH0	IMG	0	ISI channel 0
SC_R_ISI_CH1	IMG	0	ISI channel 1
SC_R_ISI_CH2	IMG	0	ISI channel 2
SC_R_ISI_CH3	IMG	0	ISI channel 3
SC_R_ISI_CH4	IMG	0	ISI channel 4
SC_R_ISI_CH5	IMG	0	ISI channel 5
SC_R_ISI_CH6	IMG	0	ISI channel 6
SC_R_ISI_CH7	IMG	0	ISI channel 7
SC_R_KPP	LSIO	0	Keypad Port (KPP)
SC_R_LVDS_0	LVDS	0	LVDS
SC_R_LVDS_0_I2C_0	LVDS	0	I2C 0 & 1
SC_R_LVDS_0_PWM_0	LVDS	0	PWM
SC_R_LVDS_1	LVDS	1	LVDS
SC_R_LVDS_1_I2C_0	LVDS	1	I2C 0 & 1
SC_R_LVDS_1_PWM_0	LVDS	1	PWM
SC_R_M4_0_I2C	M4	0	I2C
SC_R_M4_0_INTMUX	M4	0	INTMUX
SC_R_M4_0_MU_0A0	M4	0	MU 0A0
SC_R_M4_0_MU_0A1	M4	0	MU 0A1
SC_R_M4_0_MU_0A2	M4	0	MU 0A2
SC_R_M4_0_MU_0A3	M4	0	MU 0A3
SC_R_M4_0_MU_0B	M4	0	MU 0B
SC_R_M4_0_MU_1A	M4	0	MU 1A
SC_R_M4_0_PID0	M4	0	PID0
SC_R_M4_0_PID1	M4	0	PID1
SC_R_M4_0_PID2	M4	0	PID2
SC_R_M4_0_PID3	M4	0	PID3
SC_R_M4_0_PID4	M4	0	PID4
SC_R_M4_0_PIT	M4	0	PIT
SC_R_M4_0_RGPIIO	M4	0	RGPIIO
SC_R_M4_0_SEMA42	M4	0	SEMA42
SC_R_M4_0_TPM	M4	0	TPM
SC_R_M4_0_UART	M4	0	UART
SC_R_M4_1_I2C	M4	1	I2C

Resource	Subsystem	Instance	Description
SC_R_M4_1_INTMUX	M4	1	INTMUX
SC_R_M4_1_MU_0A0	M4	1	MU 0A0
SC_R_M4_1_MU_0A1	M4	1	MU 0A1
SC_R_M4_1_MU_0A2	M4	1	MU 0A2
SC_R_M4_1_MU_0A3	M4	1	MU 0A3
SC_R_M4_1_MU_0B	M4	1	MU 0B
SC_R_M4_1_MU_1A	M4	1	MU 1A
SC_R_M4_1_PID0	M4	1	PID0
SC_R_M4_1_PID1	M4	1	PID1
SC_R_M4_1_PID2	M4	1	PID2
SC_R_M4_1_PID3	M4	1	PID3
SC_R_M4_1_PID4	M4	1	PID4
SC_R_M4_1_PIT	M4	1	PIT
SC_R_M4_1_RGPIIO	M4	1	RGPIIO
SC_R_M4_1_SEMA42	M4	1	SEMA42
SC_R_M4_1_TPM	M4	1	TPM
SC_R_M4_1_UART	M4	1	UART
SC_R_MATCH_0	HSIO	0	Match 0
SC_R_MATCH_1	HSIO	0	Match 1
SC_R_MATCH_10	HSIO	0	Match 10
SC_R_MATCH_11	HSIO	0	Match 11
SC_R_MATCH_12	HSIO	0	Match 12
SC_R_MATCH_13	HSIO	0	Match 13
SC_R_MATCH_14	HSIO	0	Match 14
SC_R_MATCH_15	HSIO	0	Match 15
SC_R_MATCH_16	HSIO	0	Match 16
SC_R_MATCH_17	HSIO	0	Match 17
SC_R_MATCH_18	HSIO	0	Match 18
SC_R_MATCH_19	HSIO	0	Match 19
SC_R_MATCH_2	HSIO	0	Match 2
SC_R_MATCH_20	HSIO	0	Match 20
SC_R_MATCH_21	HSIO	0	Match 21
SC_R_MATCH_22	HSIO	0	Match 22
SC_R_MATCH_23	HSIO	0	Match 23
SC_R_MATCH_24	HSIO	0	Match 24
SC_R_MATCH_25	HSIO	0	Match 25
SC_R_MATCH_26	HSIO	0	Match 26
SC_R_MATCH_27	HSIO	0	Match 27
SC_R_MATCH_28	HSIO	0	Match 28
SC_R_MATCH_3	HSIO	0	Match 3
SC_R_MATCH_4	HSIO	0	Match 4
SC_R_MATCH_5	HSIO	0	Match 5
SC_R_MATCH_6	HSIO	0	Match 6
SC_R_MATCH_7	HSIO	0	Match 7
SC_R_MATCH_8	HSIO	0	Match 8
SC_R_MATCH_9	HSIO	0	Match 9
SC_R_MCLK_OUT_0	AUDIO	0	MCLK out 0

Resource	Subsystem	Instance	Description
SC_R_MCLK_OUT_1	AUDIO	0	MCLK out 1
SC_R_MIPI_0	MIPI	0	MIPI
SC_R_MIPI_0_I2C_0	MIPI	0	I2C 0
SC_R_MIPI_0_I2C_1	MIPI	0	I2C 1
SC_R_MIPI_0_PWM_0	MIPI	0	PWM
SC_R_MIPI_1	MIPI	1	MIPI
SC_R_MIPI_1_I2C_0	MIPI	1	I2C 0
SC_R_MIPI_1_I2C_1	MIPI	1	I2C 1
SC_R_MIPI_1_PWM_0	MIPI	1	PWM
SC_R_MJPEG_DEC_MP	IMG	0	MJPEG decoder MP
SC_R_MJPEG_DEC_S0	IMG	0	MJPEG decoder stream 0
SC_R_MJPEG_DEC_S1	IMG	0	MJPEG decoder stream 1
SC_R_MJPEG_DEC_S2	IMG	0	MJPEG decoder stream 2
SC_R_MJPEG_DEC_S3	IMG	0	MJPEG decoder stream 3
SC_R_MJPEG_ENC_MP	IMG	0	MJPEG encoder MP
SC_R_MJPEG_ENC_S0	IMG	0	MJPEG encoder stream 0
SC_R_MJPEG_ENC_S1	IMG	0	MJPEG encoder stream 1
SC_R_MJPEG_ENC_S2	IMG	0	MJPEG encoder stream 2
SC_R_MJPEG_ENC_S3	IMG	0	MJPEG encoder stream 3
SC_R_MLB_0	CONN	0	MLB
SC_R_MQS_0	AUDIO	0	MQS
SC_R_MU_0A	LSIO	0	Message Unit (MU) 0A
SC_R_MU_10A	LSIO	0	MU 10A
SC_R_MU_10B	LSIO	0	MU 10B
SC_R_MU_11A	LSIO	0	MU 11A
SC_R_MU_11B	LSIO	0	MU 11B
SC_R_MU_12A	LSIO	0	MU 12A
SC_R_MU_12B	LSIO	0	MU 12B
SC_R_MU_13A	LSIO	0	MU 13A
SC_R_MU_13B	LSIO	0	MU 13B
SC_R_MU_1A	LSIO	0	MU 1A
SC_R_MU_2A	LSIO	0	MU 2A
SC_R_MU_3A	LSIO	0	MU 3A
SC_R_MU_4A	LSIO	0	MU 4A
SC_R_MU_5A	LSIO	0	MU 5A
SC_R_MU_5B	LSIO	0	MU 5B
SC_R_MU_6A	LSIO	0	MU 6A
SC_R_MU_6B	LSIO	0	MU 6B
SC_R_MU_7A	LSIO	0	MU 7A
SC_R_MU_7B	LSIO	0	MU 7B
SC_R_MU_8A	LSIO	0	MU 8A
SC_R_MU_8B	LSIO	0	MU 8B
SC_R_MU_9A	LSIO	0	MU 9A
SC_R_MU_9B	LSIO	0	MU 9B
SC_R_NAND	CONN	0	NAND BCH/GPMI/DMA
SC_R_OCRAM	LSIO	0	OCRAM
SC_R_OTP	SC	0	OTP

Resource	Subsystem	Instance	Description
SC_R_PCIE_A	HSIO	0	PCIE A
SC_R_PCIE_B	HSIO	0	PCIE B
SC_R_PMIC_0	BOARD	0	PMIC 0
SC_R_PMIC_1	BOARD	0	PMIC 1
SC_R_PMIC_2	BOARD	0	PMIC 2
SC_R_PWM_0	LSIO	0	Pulse Width Modulator (PWM) 0
SC_R_PWM_1	LSIO	0	PWM 1
SC_R_PWM_2	LSIO	0	PWM 2
SC_R_PWM_3	LSIO	0	PWM 3
SC_R_PWM_4	LSIO	0	PWM 4
SC_R_PWM_5	LSIO	0	PWM 5
SC_R_PWM_6	LSIO	0	PWM 6
SC_R_PWM_7	LSIO	0	PWM 7
SC_R_ROM_0	LSIO	0	AP Read Only Memory (ROM)
SC_R_SAI_0	AUDIO	0	SAI 0
SC_R_SAI_1	AUDIO	0	SAI 1
SC_R_SAI_2	AUDIO	0	SAI 2
SC_R_SAI_3	AUDIO	0	SAI 3
SC_R_SAI_4	AUDIO	0	HDMI RX SAI
SC_R_SAI_5	AUDIO	0	HDMI TX SAI
SC_R_SAI_6	AUDIO	0	SAI_6
SC_R_SAI_7	AUDIO	0	SAI_7
SC_R_SATA_0	HSIO	0	SATA 0
SC_R_SC_I2C	SC	0	I2C
SC_R_SC_MU_0A0	SC	0	MU 0A0
SC_R_SC_MU_0A1	SC	0	MU 0A1
SC_R_SC_MU_0A2	SC	0	MU 0A2
SC_R_SC_MU_0A3	SC	0	MU 0A3
SC_R_SC_MU_0B	SC	0	MU 0B
SC_R_SC_MU_1A	SC	0	MU 1A
SC_R_SC_PID0	SC	0	PID0
SC_R_SC_PID1	SC	0	PID1
SC_R_SC_PID2	SC	0	PID2
SC_R_SC_PID3	SC	0	PID3
SC_R_SC_PID4	SC	0	PID4
SC_R_SC_PIT	SC	0	PIT
SC_R_SC_SEMA42	SC	0	SEMA42
SC_R_SC_TPM	SC	0	TPM
SC_R_SC_UART	SC	0	UART
SC_R_SDHC_0	CONN	0	uSDHC 1
SC_R_SDHC_1	CONN	0	uSDHC 2
SC_R_SDHC_2	CONN	0	uSDHC 3
SC_R_SECO	SC	0	SECO
SC_R_SECO_MU_2	SC	0	SECO MU 2
SC_R_SECO_MU_3	SC	0	SECO MU 3
SC_R_SECO_MU_4	SC	0	SECO MU 4
SC_R_SERDES_0	HSIO	0	PHYx2

Resource	Subsystem	Instance	Description
SC_R_SERDES_1	HSIO	0	PHYx1
SC_R_SMMU	DBLOGIC	0	SMMU
SC_R_SPDIF_0	AUDIO	0	SPDIF 0
SC_R_SPDIF_1	AUDIO	0	SPDIF 1
SC_R_SPI_0	DMA	0	SPI 0
SC_R_SPI_1	DMA	0	SPI 1
SC_R_SPI_2	DMA	0	SPI 2
SC_R_SPI_3	DMA	0	SPI 3
SC_R_SYSCNT_CMP	SC	0	SYSCNT CMP
SC_R_SYSCNT_RD	SC	0	SYSCNT RD
SC_R_SYSTEM	SC	0	System
SC_R_UART_0	DMA	0	UART 0
SC_R_UART_1	DMA	0	UART 1
SC_R_UART_2	DMA	0	UART 2
SC_R_UART_3	DMA	0	UART 3
SC_R_UART_4	DMA	0	UART 4
SC_R_USB_0	CONN	0	USB2.0 OTG
SC_R_USB_0_PHY	CONN	0	USB2.0 OTG phy
SC_R_USB_1	CONN	0	USB2.0 HSIC
SC_R_USB_2	CONN	0	USB3.0
SC_R_USB_2_PHY	CONN	0	USB3.0 phy
SC_R_VPU	VPU	0	VPU (XMEM peripheral)
SC_R_VPU_DEC_0	VPU	0	Decoder
SC_R_VPU_ENC_0	VPU	0	Encoder
SC_R_VPU_PID0	VPU	0	XMEM master ID=0
SC_R_VPU_PID1	VPU	0	XMEM master ID=1
SC_R_VPU_PID2	VPU	0	XMEM master ID=2
SC_R_VPU_PID3	VPU	0	XMEM master ID=3
SC_R_VPU_PID4	VPU	0	XMEM master ID=4
SC_R_VPU_PID5	VPU	0	XMEM master ID=5
SC_R_VPU_PID6	VPU	0	XMEM master ID=6
SC_R_VPU_PID7	VPU	0	XMEM master ID=7
SC_R_VPU_UART	VPU	0	UART

Chapter 5

Clock List

The table below lists the clocks/PLLs available in the system (SoC specific). Resource and Clock are parameters to several PM API calls. Set=Y indicates the clock/PLL is not shared and the rate can be set via [sc_pm_set_clock_rate\(\)](#). Enable=Y indicates the clock is not auto gated and must be enabled via [sc_pm_clock_enable\(\)](#).

See also

[\(SVC\) Power Management Service](#)

Resource	Clock	Default Rate	Set	Enable	Description
SC_R_A53	SC_PM_CLK_CPU	1200MHZ	Y		CPU
SC_R_A72	SC_PM_CLK_CPU	1600MHZ	Y		CPU
SC_R_ADC_0	SC_PM_CLK_PER		Y	Y	ADC 0 peripheral
SC_R_ADC_1	SC_PM_CLK_PER		Y	Y	ADC 1 peripheral
SC_R_AUDIO_PLL_0	SC_PM_CLK_MISC0		Y	Y	Audio PLL div 0
SC_R_AUDIO_PLL_0	SC_PM_CLK_MISC1		Y	Y	Audio rec 0
SC_R_AUDIO_PLL_0	SC_PM_CLK_PLL		Y	Y	User programmable PLL
SC_R_AUDIO_PLL_1	SC_PM_CLK_MISC0		Y	Y	Audio PLL div 1
SC_R_AUDIO_PLL_1	SC_PM_CLK_MISC1		Y	Y	Audio rec 1
SC_R_AUDIO_PLL_1	SC_PM_CLK_PLL		Y	Y	User programmable PLL
SC_R_CAN_0	SC_PM_CLK_PER		Y	Y	CAN 0 peripheral
SC_R_CAN_1	SC_PM_CLK_PER		Y	Y	CAN 1 peripheral
SC_R_CAN_2	SC_PM_CLK_PER		Y	Y	CAN 2 peripheral
SC_R_CSI_0	SC_PM_CLK_MISC		Y	Y	MIPI escape
SC_R_CSI_0	SC_PM_CLK_PER		Y	Y	MIPI core
SC_R_CSI_0_I2C_0	SC_PM_CLK_PER		Y	Y	I2C baud
SC_R_CSI_0_PWM_0	SC_PM_CLK_PER		Y	Y	PWM high
SC_R_CSI_1	SC_PM_CLK_MISC		Y	Y	MIPI escape
SC_R_CSI_1	SC_PM_CLK_PER		Y	Y	MIPI core
SC_R_CSI_1_I2C_0	SC_PM_CLK_PER		Y	Y	I2C baud
SC_R_CSI_1_PWM_0	SC_PM_CLK_PER		Y	Y	PWM high
SC_R_DC_0	SC_PM_CLK_MISC0		Y	Y	Display 0
SC_R_DC_0	SC_PM_CLK_MISC1		Y	Y	Display 1

Resource	Clock	Default Rate	Set	Enable	Description
SC_R_DC_0_PLL_0	SC_PM_CLK_PLL		Y	Y	User programmable PLL
SC_R_DC_0_PLL_1	SC_PM_CLK_PLL		Y	Y	User programmable PLL
SC_R_DC_0_VIDEO0	SC_PM_CLK_MISC	24MHZ	Y	Y	Bypass 0
SC_R_DC_0_VIDEO1	SC_PM_CLK_MISC	24MHZ	Y	Y	Bypass 1
SC_R_DC_1	SC_PM_CLK_MISC0		Y	Y	Display 0
SC_R_DC_1	SC_PM_CLK_MISC1		Y	Y	Display 1
SC_R_DC_1_PLL_0	SC_PM_CLK_PLL		Y	Y	User programmable PLL
SC_R_DC_1_PLL_1	SC_PM_CLK_PLL		Y	Y	User programmable PLL
SC_R_DC_1_VIDEO0	SC_PM_CLK_MISC	24MHZ	Y	Y	Bypass 0
SC_R_DC_1_VIDEO1	SC_PM_CLK_MISC	24MHZ	Y	Y	Bypass 1
SC_R_DRC_0	SC_PM_CLK_SLV_B↔US	800MHZ	Y		DRC root
SC_R_DRC_1	SC_PM_CLK_SLV_B↔US	800MHZ	Y		DRC root
SC_R_EMVSIM_0	SC_PM_CLK_PER		Y	Y	EMV_SIM 0 peripheral
SC_R_EMVSIM_1	SC_PM_CLK_PER		Y	Y	EMV_SIM 1 peripheral
SC_R_ENET_0	SC_PM_CLK_BYPASS	24MHZ	Y	Y	ENET 0 bypass
SC_R_ENET_0	SC_PM_CLK_MISC0	24MHZ	Y	Y	ENET 0 RXCLK/RGM↔II_CLK
SC_R_ENET_0	SC_PM_CLK_PER	125MHZ	Y	Y	ENET 0 PER_CLK/RE↔F_CLK/CLK
SC_R_ENET_1	SC_PM_CLK_BYPASS	24MHZ	Y	Y	ENET 1 bypass
SC_R_ENET_1	SC_PM_CLK_MISC0	24MHZ	Y	Y	ENET 1 RXCLK/RGM↔II_CLK
SC_R_ENET_1	SC_PM_CLK_PER	125MHZ	Y	Y	ENET 1 PER_CLK/RE↔F_CLK/CLK
SC_R_FSPI_0	SC_PM_CLK_PER		Y	Y	FSPI 0 baud
SC_R_FSPI_1	SC_PM_CLK_PER		Y	Y	FSPI 1 baud
SC_R_FTM_0	SC_PM_CLK_PER		Y	Y	FTM 0 peripheral
SC_R_FTM_1	SC_PM_CLK_PER		Y	Y	FTM 1 peripheral
SC_R_GPT_0	SC_PM_CLK_PER		Y	Y	GPT 0 peripheral
SC_R_GPT_1	SC_PM_CLK_PER		Y	Y	GPT 1 peripheral
SC_R_GPT_2	SC_PM_CLK_PER		Y	Y	GPT 2 peripheral
SC_R_GPT_3	SC_PM_CLK_PER		Y	Y	GPT 3 peripheral
SC_R_GPT_4	SC_PM_CLK_PER		Y	Y	GPT 4 peripheral
SC_R_GPU_0_PID0	SC_PM_CLK_MISC		Y	Y	Shader
SC_R_GPU_0_PID0	SC_PM_CLK_PER		Y	Y	GPU
SC_R_GPU_1_PID0	SC_PM_CLK_MISC		Y	Y	Shader
SC_R_GPU_1_PID0	SC_PM_CLK_PER		Y	Y	GPU
SC_R_HDMI	SC_PM_CLK_MISC0		Y	Y	Pixel link mux
SC_R_HDMI	SC_PM_CLK_MISC1		Y	Y	Pixel link
SC_R_HDMI	SC_PM_CLK_MISC2	135MHZ	Y	Y	HDP core
SC_R_HDMI	SC_PM_CLK_MISC3		Y	Y	Pixel
SC_R_HDMI	SC_PM_CLK_MISC4	84MHZ	Y	Y	Bus
SC_R_HDMI_I2C_0	SC_PM_CLK_MISC1	84MHZ	Y	Y	Bus
SC_R_HDMI_I2C_0	SC_PM_CLK_MISC2	24MHZ	Y	Y	I2C baud
SC_R_HDMI_I2S	SC_PM_CLK_MISC0		Y	Y	I2S

Resource	Clock	Default Rate	Set	Enable	Description
SC_R_HDMI_I2S	SC_PM_CLK_MISC1	84MHZ	Y	Y	Bus
SC_R_HDMI_I2S	SC_PM_CLK_MISC4		Y	Y	I2S bypass
SC_R_HDMI_PLL_0	SC_PM_CLK_PLL	675MHZ	Y	Y	User programmable PLL
SC_R_HDMI_PLL_1	SC_PM_CLK_PLL		Y	Y	User programmable PLL
SC_R_HDMI_RX	SC_PM_CLK_MISC0		Y	Y	SPDIF
SC_R_HDMI_RX	SC_PM_CLK_MISC1		Y	Y	HDP ref
SC_R_HDMI_RX	SC_PM_CLK_MISC2	200MHZ	Y		HDP core
SC_R_HDMI_RX	SC_PM_CLK_MISC3		Y	Y	Pixel
SC_R_HDMI_RX	SC_PM_CLK_MISC4		Y	Y	I2S
SC_R_HDMI_RX_BY↔ PASS	SC_PM_CLK_MISC0		Y	Y	I2S bypass
SC_R_HDMI_RX_BY↔ PASS	SC_PM_CLK_MISC1		Y	Y	SPDIF bypass
SC_R_HDMI_RX_BY↔ PASS	SC_PM_CLK_MISC2		Y	Y	HDMI bypass
SC_R_HDMI_RX_I2C↔ _0	SC_PM_CLK_MISC2	24MHZ	Y	Y	I2C baud
SC_R_HDMI_RX_P↔ WM_0	SC_PM_CLK_MISC2		Y	Y	PWM highfreq
SC_R_I2C_0	SC_PM_CLK_PER		Y	Y	I2C 0 baud
SC_R_I2C_1	SC_PM_CLK_PER		Y	Y	I2C 1 baud
SC_R_I2C_2	SC_PM_CLK_PER		Y	Y	I2C 2 baud
SC_R_I2C_3	SC_PM_CLK_PER		Y	Y	I2C 3 baud
SC_R_I2C_4	SC_PM_CLK_PER		Y	Y	I2C 4 baud
SC_R_LVDS_0	SC_PM_CLK_BYPASS		Y	Y	Bypass
SC_R_LVDS_0	SC_PM_CLK_PER		Y	Y	Pixel
SC_R_LVDS_0	SC_PM_CLK_PHY		Y	Y	Phy
SC_R_LVDS_0_I2C_0	SC_PM_CLK_PER	24MHZ	Y	Y	I2C 0 & 1 baud
SC_R_LVDS_0_PW↔ M_0	SC_PM_CLK_PER	24MHZ	Y	Y	PWM baud
SC_R_LVDS_1	SC_PM_CLK_BYPASS		Y	Y	Bypass
SC_R_LVDS_1	SC_PM_CLK_PER		Y	Y	Pixel
SC_R_LVDS_1	SC_PM_CLK_PHY		Y	Y	Phy
SC_R_LVDS_1_I2C_0	SC_PM_CLK_PER	24MHZ	Y	Y	I2C 0 & 1 baud
SC_R_LVDS_1_PW↔ M_0	SC_PM_CLK_PER	24MHZ	Y	Y	PWM baud
SC_R_M4_0_I2C	SC_PM_CLK_PER		Y	Y	I2C baud
SC_R_M4_0_PID0	SC_PM_CLK_CPU	264MHZ	Y		System
SC_R_M4_0_PIT	SC_PM_CLK_PER		Y	Y	LPIT peripheral
SC_R_M4_0_TPM	SC_PM_CLK_PER		Y	Y	TPM peripheral
SC_R_M4_0_UART	SC_PM_CLK_PER		Y	Y	UART baud
SC_R_M4_1_I2C	SC_PM_CLK_PER		Y	Y	I2C baud
SC_R_M4_1_PID0	SC_PM_CLK_CPU	264MHZ	Y		System
SC_R_M4_1_PIT	SC_PM_CLK_PER		Y	Y	LPIT peripheral
SC_R_M4_1_TPM	SC_PM_CLK_PER		Y	Y	TPM peripheral
SC_R_M4_1_UART	SC_PM_CLK_PER		Y	Y	UART baud
SC_R_MIPI_0	SC_PM_CLK_BYPASS	24MHZ	Y	Y	Bypass

Resource	Clock	Default Rate	Set	Enable	Description
SC_R_MIPI_0	SC_PM_CLK_MST_B↔ US		Y	Y	DSI tx escape
SC_R_MIPI_0	SC_PM_CLK_PER	24MHZ	Y	Y	DPI (pixel)
SC_R_MIPI_0	SC_PM_CLK_PHY	27MHZ	Y		DPHY PLL ref
SC_R_MIPI_0	SC_PM_CLK_SLV_B↔ US		Y	Y	DSI rx escape
SC_R_MIPI_0_I2C_0	SC_PM_CLK_PER	40MHZ	Y	Y	I2C 0 baud
SC_R_MIPI_0_I2C_1	SC_PM_CLK_PER	40MHZ	Y	Y	I2C 1 baud
SC_R_MIPI_0_PWM↔ _0	SC_PM_CLK_PER	24MHZ	Y	Y	PWM baud
SC_R_MIPI_1	SC_PM_CLK_BYPASS	24MHZ	Y	Y	Bypass
SC_R_MIPI_1	SC_PM_CLK_MST_B↔ US		Y	Y	DSI tx escape
SC_R_MIPI_1	SC_PM_CLK_PER	24MHZ	Y	Y	DPI (pixel)
SC_R_MIPI_1	SC_PM_CLK_PHY	27MHZ	Y		DPHY PLL ref
SC_R_MIPI_1	SC_PM_CLK_SLV_B↔ US		Y	Y	DSI rx escape
SC_R_MIPI_1_I2C_0	SC_PM_CLK_PER	40MHZ	Y	Y	I2C 0 baud
SC_R_MIPI_1_I2C_1	SC_PM_CLK_PER	40MHZ	Y	Y	I2C 1 baud
SC_R_MIPI_1_PWM↔ _0	SC_PM_CLK_PER	24MHZ	Y	Y	PWM baud
SC_R_NAND	SC_PM_CLK_MST_B↔ US	500MHZ	Y	Y	NAND GPMI
SC_R_NAND	SC_PM_CLK_PER	400MHZ	Y	Y	NAND BCH
SC_R_PWM_0	SC_PM_CLK_PER		Y	Y	PWM 0 peripheral
SC_R_PWM_1	SC_PM_CLK_PER		Y	Y	PWM 1 peripheral
SC_R_PWM_2	SC_PM_CLK_PER		Y	Y	PWM 2 peripheral
SC_R_PWM_3	SC_PM_CLK_PER		Y	Y	PWM 3 peripheral
SC_R_PWM_4	SC_PM_CLK_PER		Y	Y	PWM 4 peripheral
SC_R_PWM_5	SC_PM_CLK_PER		Y	Y	PWM 5 peripheral
SC_R_PWM_6	SC_PM_CLK_PER		Y	Y	PWM 6 peripheral
SC_R_PWM_7	SC_PM_CLK_PER		Y	Y	PWM 7 peripheral
SC_R_SC_I2C	SC_PM_CLK_PER	24MHZ	Y		I2C baud
SC_R_SC_PID0	SC_PM_CLK_CPU	264MHZ	N		System
SC_R_SC_PIT	SC_PM_CLK_PER	8MHZ	Y		LPIT peripheral
SC_R_SC_TPM	SC_PM_CLK_PER		Y	Y	TPM peripheral
SC_R_SC_UART	SC_PM_CLK_PER	24MHZ	Y		UART baud
SC_R_SDHC_0	SC_PM_CLK_PER	396MHZ	Y	Y	uSDHC 0 peripheral
SC_R_SDHC_1	SC_PM_CLK_PER	396MHZ	Y	Y	uSDHC 1 peripheral
SC_R_SDHC_2	SC_PM_CLK_PER	396MHZ	Y	Y	uSDHC 2 peripheral
SC_R_SPI_0	SC_PM_CLK_PER		Y	Y	SPI 0 baud
SC_R_SPI_1	SC_PM_CLK_PER		Y	Y	SPI 1 baud
SC_R_SPI_2	SC_PM_CLK_PER		Y	Y	SPI 2 baud
SC_R_SPI_3	SC_PM_CLK_PER		Y	Y	SPI 3 baud
SC_R_UART_0	SC_PM_CLK_PER		Y	Y	UART 0 baud
SC_R_UART_1	SC_PM_CLK_PER		Y	Y	UART 1 baud
SC_R_UART_2	SC_PM_CLK_PER		Y	Y	UART 2 baud

Resource	Clock	Default Rate	Set	Enable	Description
SC_R_UART_3	SC_PM_CLK_PER		Y	Y	UART 3 baud
SC_R_UART_4	SC_PM_CLK_PER		Y	Y	UART 4 baud
SC_R_USB_2	SC_PM_CLK_MISC	12MHZ	Y	Y	USB 2 lpm
SC_R_USB_2	SC_PM_CLK_MST_B↔ US	500MHZ	Y	Y	USB 2 bus
SC_R_USB_2	SC_PM_CLK_PER	125MHZ	Y	Y	USB 2
SC_R_VPU	SC_PM_CLK_MST_B↔ US		Y	Y	CABAC
SC_R_VPU	SC_PM_CLK_PER		Y	Y	XUVI
SC_R_VPU	SC_PM_CLK_SLV_B↔ US		Y	Y	DDR
SC_R_VPU_UART	SC_PM_CLK_PER		Y	Y	UART baud

Chapter 6

Control List

The table below lists the miscellaneous controls available in the system (SoC specific). Resource and Control are parameters to several MISC API calls. Set=Y indicates the control can be written via [sc_misc_set_control\(\)](#). All controls can be read via [sc_misc_get_control\(\)](#).

See also

[\(SVC\) Miscellaneous Service](#)

Resource	Control	Width	Set	Description
SC_R_A53	SC_C_TEMP	16		Temperature sensor value
SC_R_A53	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_A53	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_A72	SC_C_TEMP	16		Temperature sensor value
SC_R_A72	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_A72	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_CAN_0	SC_C_IPG_STOP	1	Y	CAN0 IPG_STOP
SC_R_CAN_0	SC_C_IPG_STOP_ACK	1	Y	CAN0 IPG_STOP_ACK
SC_R_CAN_1	SC_C_IPG_STOP	1	Y	CAN1 IPG_STOP
SC_R_CAN_1	SC_C_IPG_STOP_ACK	1	Y	CAN1 IPG_STOP_ACK
SC_R_CAN_2	SC_C_IPG_STOP	1	Y	CAN2 IPG_STOP
SC_R_CAN_2	SC_C_IPG_STOP_ACK	1	Y	CAN2 IPG_STOP_ACK
SC_R_CSI_0	SC_C_CALIB1	2	Y	MIPI CSI LP-RX threshold voltage
SC_R_CSI_0	SC_C_CALIB2	2	Y	MIPI CSI LP-CD threshold voltage
SC_R_CSI_0	SC_C_MIPI_RESET	1	Y	MIPI CSI reset_n, escape and UI resets
SC_R_CSI_1	SC_C_CALIB1	2	Y	MIPI CSI LP-RX threshold voltage
SC_R_CSI_1	SC_C_CALIB2	2	Y	MIPI CSI LP-CD threshold voltage
SC_R_CSI_1	SC_C_MIPI_RESET	1	Y	MIPI CSI reset_n, escape and UI resets

Resource	Control	Width	Set	Description
SC_R_DC_0	SC_C_KACHUNK_CNT	16	Y	CTX kachunk count
SC_R_DC_0	SC_C_MODE	1	Y	Sync mode
SC_R_DC_0	SC_C_PANIC	2	Y	Panic switch
SC_R_DC_0	SC_C_PXL_LINK_MST1_ADDR	2	Y	Master 1 pixel link address
SC_R_DC_0	SC_C_PXL_LINK_MST1_ENB	1	Y	Master 1 pixel link enable
SC_R_DC_0	SC_C_PXL_LINK_MST1_VLD	1	Y	Master 1 pixel link valid
SC_R_DC_0	SC_C_PXL_LINK_MST2_ADDR	2	Y	Master 2 pixel link address
SC_R_DC_0	SC_C_PXL_LINK_MST2_ENB	1	Y	Master 2 pixel link enable
SC_R_DC_0	SC_C_PXL_LINK_MST2_VLD	1	Y	Master 2 pixel link valid
SC_R_DC_0	SC_C_PXL_LINK_MST_ENB	2	Y	Combined master pixel link enable
SC_R_DC_0	SC_C_PXL_LINK_MST_VLD	2	Y	Combined master pixel link valid
SC_R_DC_0	SC_C_RST0	1	Y	Display Stream 0 reset
SC_R_DC_0	SC_C_RST1	1	Y	Display Stream 1 reset
SC_R_DC_0	SC_C_SYNC_CTRL	2	Y	PL sync ctrl 0 & 1
SC_R_DC_0	SC_C_SYNC_CTRL0	1	Y	PL sync ctrl 0
SC_R_DC_0	SC_C_SYNC_CTRL1	1	Y	PL sync ctrl 1
SC_R_DC_0_BLIT0	SC_C_SEL0	1	Y	Blit channel 0 select
SC_R_DC_0_FRAC0	SC_C_KACHUNK_SEL	1	Y	FRAC0 kachunk sel
SC_R_DC_0_VIDEO0	SC_C_KACHUNK_SEL	1	Y	VID0 kachunk sel
SC_R_DC_0_VIDEO1	SC_C_KACHUNK_SEL	1	Y	VID1 kachunk sel
SC_R_DC_0_WARP	SC_C_KACHUNK_SEL	1	Y	WARP kachunk sel
SC_R_DC_1	SC_C_KACHUNK_CNT	16	Y	CTX kachunk count
SC_R_DC_1	SC_C_MODE	1	Y	Sync mode
SC_R_DC_1	SC_C_PANIC	2	Y	Panic switch
SC_R_DC_1	SC_C_PXL_LINK_MST1_ADDR	2	Y	Master 1 pixel link address
SC_R_DC_1	SC_C_PXL_LINK_MST1_ENB	1	Y	Master 1 pixel link enable
SC_R_DC_1	SC_C_PXL_LINK_MST1_VLD	1	Y	Master 1 pixel link valid
SC_R_DC_1	SC_C_PXL_LINK_MST2_ADDR	2	Y	Master 2 pixel link address
SC_R_DC_1	SC_C_PXL_LINK_MST2_ENB	1	Y	Master 2 pixel link enable
SC_R_DC_1	SC_C_PXL_LINK_MST2_VLD	1	Y	Master 2 pixel link valid
SC_R_DC_1	SC_C_PXL_LINK_MST_ENB	2	Y	Combined master pixel link enable
SC_R_DC_1	SC_C_PXL_LINK_MST_VLD	2	Y	Combined master pixel link valid
SC_R_DC_1	SC_C_RST0	1	Y	Display Stream 0 reset
SC_R_DC_1	SC_C_RST1	1	Y	Display Stream 1 reset
SC_R_DC_1	SC_C_SYNC_CTRL	2	Y	PL sync ctrl 0 & 1
SC_R_DC_1	SC_C_SYNC_CTRL0	1	Y	PL sync ctrl 0
SC_R_DC_1	SC_C_SYNC_CTRL1	1	Y	PL sync ctrl 1
SC_R_DC_1_BLIT0	SC_C_SEL0	1	Y	Blit channel 0 select
SC_R_DC_1_FRAC0	SC_C_KACHUNK_SEL	1	Y	FRAC0 kachunk sel
SC_R_DC_1_VIDEO0	SC_C_KACHUNK_SEL	1	Y	VID0 kachunk sel
SC_R_DC_1_VIDEO1	SC_C_KACHUNK_SEL	1	Y	VID1 kachunk sel
SC_R_DC_1_WARP	SC_C_KACHUNK_SEL	1	Y	WARP kachunk sel
SC_R_DRC_0	SC_C_TEMP	16		Temperature sensor value
SC_R_DRC_0	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value

Resource	Control	Width	Set	Description
SC_R_DRC_0	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_DRC_1	SC_C_TEMP	16		Temperature sensor value
SC_R_DRC_1	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_DRC_1	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_ENET_0	SC_C_CLKDIV	1	Y	ENET1 divided clock
SC_R_ENET_0	SC_C_DISABLE_125	1	Y	ENET1 125/25MHz clock disable
SC_R_ENET_0	SC_C_DISABLE_50	1	Y	ENET1 50MHz clock disable
SC_R_ENET_0	SC_C_IPG_STOP	1	Y	ENET1 IPG stop mode
SC_R_ENET_0	SC_C_SEL_125	1	Y	ENET1 125/25MHz ref clk sel
SC_R_ENET_0	SC_C_TXCLK	1	Y	ENET1 TXCLK selection
SC_R_ENET_1	SC_C_CLKDIV	1	Y	ENET2 divided clock
SC_R_ENET_1	SC_C_DISABLE_125	1	Y	ENET2 125/25MHz clock disable
SC_R_ENET_1	SC_C_DISABLE_50	1	Y	ENET2 50MHz clock disable
SC_R_ENET_1	SC_C_IPG_STOP	1	Y	ENET2 IPG stop mode
SC_R_ENET_1	SC_C_SEL_125	1	Y	ENET2 125/25MHz ref clk sel
SC_R_ENET_1	SC_C_TXCLK	1	Y	ENET2 TXCLK selection
SC_R_GPU_0_PID0	SC_C_ID	4	Y	Identifier
SC_R_GPU_0_PID0	SC_C_SINGLE_MODE	1	Y	Force to single mode
SC_R_GPU_0_PID0	SC_C_TEMP	16		Temperature sensor value
SC_R_GPU_0_PID0	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_GPU_0_PID0	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_GPU_1_PID0	SC_C_ID	4	Y	Identifier
SC_R_GPU_1_PID0	SC_C_SINGLE_MODE	1	Y	Force to single mode
SC_R_GPU_1_PID0	SC_C_TEMP	16		Temperature sensor value
SC_R_GPU_1_PID0	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_GPU_1_PID0	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_HDMI	SC_C_MODE	1U	Y	0 = HDMI, 1 = DP
SC_R_HDMI	SC_C_PHY_RESET	1U	Y	HDMI PHY reset
SC_R_HDMI_RX	SC_C_PHY_RESET	1	Y	HDMI PHY reset
SC_R_LVDS_0	SC_C_PXL_LINK_RATE_CORRECTION	2	Y	
SC_R_LVDS_1	SC_C_PXL_LINK_RATE_CORRECTION	2	Y	
SC_R_MIPI_0	SC_C_DPI_RESET	1	Y	MIPI DSI DPI reset
SC_R_MIPI_0	SC_C_MIPI_RESET	1	Y	MIPI DSI escape reset
SC_R_MIPI_0	SC_C_PHY_RESET	1	Y	MIPI DSI byte reset (PHY reset)
SC_R_MIPI_1	SC_C_DPI_RESET	1	Y	MIPI DSI DPI reset
SC_R_MIPI_1	SC_C_MIPI_RESET	1	Y	MIPI DSI escape reset
SC_R_MIPI_1	SC_C_PHY_RESET	1	Y	MIPI DSI byte reset (PHY reset)
SC_R_MLB_0	SC_C_CLKDIV	3	Y	MLB PLL Multiplier (0 - refclk, 1, 2 and 4)

Resource	Control	Width	Set	Description
SC_R_MLB_0	SC_C_MODE	1	Y	0 - disable MLB PLL, 1 - Enable MLB PLL
SC_R_PMIC_0	SC_C_TEMP	8		Temperature sensor temp
SC_R_PMIC_0	SC_C_TEMP_HI	8	Y	Temperature sensor high limit alarm temp
SC_R_PMIC_1	SC_C_TEMP	8		Temperature sensor temp
SC_R_PMIC_1	SC_C_TEMP_HI	8	Y	Temperature sensor high limit alarm temp
SC_R_PMIC_2	SC_C_TEMP	8		Temperature sensor temp
SC_R_PMIC_2	SC_C_TEMP_HI	8	Y	Temperature sensor high limit alarm temp
SC_R_SDHC_0	SC_C_VOLTAGE	1	Y	uSDHC1 IO voltage
SC_R_SDHC_1	SC_C_VOLTAGE	1	Y	uSDHC2 IO voltage
SC_R_SDHC_2	SC_C_VOLTAGE	1	Y	uSDHC3 IO voltage
SC_R_SYSTEM	SC_C_ID	9		Chip ID and revision value
SC_R_SYSTEM	SC_C_MODE	8		Boot mode pads
SC_R_SYSTEM	SC_C_TEMP	16		Temperature sensor value
SC_R_SYSTEM	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_SYSTEM	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_VPU	SC_C_TEMP	16		Temperature sensor value
SC_R_VPU	SC_C_TEMP_HI	16	Y	Temperature sensor high limit alarm value
SC_R_VPU	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value

Chapter 7

Pad List

The table below lists the pads available in the system (SoC specific). Pad is a parameter to several PAD API calls.

Note, if two pads are configured for the same IP signal, one is successful and the other will not get the signal. This prioritization is done in HW and some pads have a default mux setting to a signal that another pad can be set to. If SW tries to set the other pad to this setting, it may not make a connection if that pad is lower priority in the HW than the one that has this setting due to default. To resolve this, the SW has to move the conflicting pad to something other than the offending signal. This isn't always possible if the offending pad is owned by another partition. As a result, the SCFW changes the mux configuration of all the pads that could be in conflict to GPIO as those are never in conflict. Refer to the SC_PAD_INIT_INIT define in the SoC-specific soc.h for a list of these pads.

See also

[\(SVC\) Pad Service](#)

Pad	Mux Options
SC_P_ADC_IN0	DMA.ADC0.IN0, LSIO.KPP0.COL0, LSIO.GPIO3.IO18
SC_P_ADC_IN1	DMA.ADC0.IN1, LSIO.KPP0.COL1, LSIO.GPIO3.IO19
SC_P_ADC_IN2	DMA.ADC0.IN2, LSIO.KPP0.COL2, LSIO.GPIO3.IO20
SC_P_ADC_IN3	DMA.ADC0.IN3, DMA.SPI1.SCK, LSIO.KPP0.COL3, LSIO.GPIO3.IO21
SC_P_ADC_IN4	DMA.ADC1.IN0, DMA.SPI1.SDO, LSIO.KPP0.ROW0, LSIO.GPIO3.IO22
SC_P_ADC_IN5	DMA.ADC1.IN1, DMA.SPI1.SDI, LSIO.KPP0.ROW1, LSIO.GPIO3.IO23
SC_P_ADC_IN6	DMA.ADC1.IN2, DMA.SPI1.CS0, LSIO.KPP0.ROW2, LSIO.GPIO3.IO24
SC_P_ADC_IN7	DMA.ADC1.IN3, DMA.SPI1.CS1, LSIO.KPP0.ROW3, LSIO.GPIO3.IO25
SC_P_CALIBRATION_0_HSIC	
SC_P_CALIBRATION_1_HSIC	
SC_P_COMP_CTL_GPIO_1V8_3V3_ENET_ENETA	
SC_P_COMP_CTL_GPIO_1V8_3V3_ENET_ENETB	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIOCT	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIOLH	

Pad	Mux Options
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIOLHT	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHB	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHC	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHT	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIOTHR	
SC_P_COMP_CTL_GPIO_1V8_3V3_LVDSGPIO	
SC_P_COMP_CTL_GPIO_1V8_3V3_MIPIDSIGPIO	
SC_P_COMP_CTL_GPIO_1V8_3V3_PCIESEP	
SC_P_COMP_CTL_GPIO_1V8_3V3_QSPI0	
SC_P_COMP_CTL_GPIO_1V8_3V3_QSPI1	
SC_P_COMP_CTL_GPIO_1V8_3V3_SD1FIX	
SC_P_COMP_CTL_GPIO_1V8_3V3_SIM	
SC_P_COMP_CTL_GPIO_1V8_3V3_VSEL2	
SC_P_COMP_CTL_GPIO_1V8_3V3_VSEL3	
SC_P_COMP_CTL_GPIO_1V8_3V3_VSELSEP	
SC_P_COMP_CTL_GPIO_3V3_HDMIGPIO	
SC_P_COMP_CTL_GPIO_3V3_USB3IO	
SC_P_CTL_NAND_DQS_P_N	
SC_P_CTL_NAND_RE_P_N	
SC_P_EMMC0_CLK	CONN.EMMC0.CLK, CONN.NAND.READY_B
SC_P_EMMC0_CMD	CONN.EMMC0.CMD, CONN.NAND.DQS, AUD.MQS.R, LSIO.GPIO5.IO03
SC_P_EMMC0_DATA0	CONN.EMMC0.DATA0, CONN.NAND.DATA00, LSI↔O.GPIO5.IO04
SC_P_EMMC0_DATA1	CONN.EMMC0.DATA1, CONN.NAND.DATA01, LSI↔O.GPIO5.IO05
SC_P_EMMC0_DATA2	CONN.EMMC0.DATA2, CONN.NAND.DATA02, LSI↔O.GPIO5.IO06
SC_P_EMMC0_DATA3	CONN.EMMC0.DATA3, CONN.NAND.DATA03, LSI↔O.GPIO5.IO07
SC_P_EMMC0_DATA4	CONN.EMMC0.DATA4, CONN.NAND.DATA04, LSI↔O.GPIO5.IO08
SC_P_EMMC0_DATA5	CONN.EMMC0.DATA5, CONN.NAND.DATA05, LSI↔O.GPIO5.IO09
SC_P_EMMC0_DATA6	CONN.EMMC0.DATA6, CONN.NAND.DATA06, LSI↔O.GPIO5.IO10
SC_P_EMMC0_DATA7	CONN.EMMC0.DATA7, CONN.NAND.DATA07, LSI↔O.GPIO5.IO11
SC_P_EMMC0_RESET_B	CONN.EMMC0.RESET_B, CONN.NAND.WP_B, CO↔NN.USDHC1.VSELECT, LSIO.GPIO5.IO13
SC_P_EMMC0_STROBE	CONN.EMMC0.STROBE, CONN.NAND.CLE, LSIO.G↔PIO5.IO12
SC_P_ENET0_MDC	CONN.ENET0.MDC, DMA.I2C4.SCL, LSIO.GPIO4.IO14
SC_P_ENET0_MDIO	CONN.ENET0.MDIO, DMA.I2C4.SDA, LSIO.GPIO4.I↔O13
SC_P_ENET0_REFCLK_125M_25M	CONN.ENET0.REFCLK_125M_25M, CONN.ENET0.↔PPS, LSIO.GPIO4.IO15
SC_P_ENET0_RGMII_RXC	CONN.ENET0.RGMII_RXC, DMA.UART3.CTS_B, VP↔U.TSI_S1.DATA, LSIO.GPIO6.IO04

Pad	Mux Options
SC_P_ENET0_RGMII_RXD0	CONN.ENET0.RGMII_RXD0, VPU.TSI_S0.SYNC, LSIO.GPIO6.IO06
SC_P_ENET0_RGMII_RXD1	CONN.ENET0.RGMII_RXD1, VPU.TSI_S0.DATA, LSIO.GPIO6.IO07
SC_P_ENET0_RGMII_RXD2	CONN.ENET0.RGMII_RXD2, CONN.ENET0.RMII_RX_ER, VPU.TSI_S0.CLK, LSIO.GPIO6.IO08
SC_P_ENET0_RGMII_RXD3	CONN.ENET0.RGMII_RXD3, DMA.UART3.RX, VPU.TSI_S1.CLK, LSIO.GPIO6.IO09
SC_P_ENET0_RGMII_RX_CTL	CONN.ENET0.RGMII_RX_CTL, VPU.TSI_S0.VID, LSIO.GPIO6.IO05
SC_P_ENET0_RGMII_TXC	CONN.ENET0.RGMII_TXC, CONN.ENET0.RCLK50M_OUT, CONN.ENET0.RCLK50M_IN, LSIO.GPIO5.IO30
SC_P_ENET0_RGMII_TXD0	CONN.ENET0.RGMII_TXD0, LSIO.GPIO6.IO00
SC_P_ENET0_RGMII_TXD1	CONN.ENET0.RGMII_TXD1, LSIO.GPIO6.IO01
SC_P_ENET0_RGMII_TXD2	CONN.ENET0.RGMII_TXD2, DMA.UART3.TX, VPU.TSI_S1.VID, LSIO.GPIO6.IO02
SC_P_ENET0_RGMII_TXD3	CONN.ENET0.RGMII_TXD3, DMA.UART3.RTS_B, VPU.TSI_S1.SYNC, LSIO.GPIO6.IO03
SC_P_ENET0_RGMII_TX_CTL	CONN.ENET0.RGMII_TX_CTL, LSIO.GPIO5.IO31
SC_P_ENET1_MDC	CONN.ENET1.MDC, DMA.I2C4.SCL, LSIO.GPIO4.IO18
SC_P_ENET1_MDIO	CONN.ENET1.MDIO, DMA.I2C4.SDA, LSIO.GPIO4.IO17
SC_P_ENET1_REFCLK_125M_25M	CONN.ENET1.REFCLK_125M_25M, CONN.ENET1.PPS, LSIO.GPIO4.IO16
SC_P_ENET1_RGMII_RXC	CONN.ENET1.RGMII_RXC, DMA.UART3.CTS_B, VPU.TSI_S1.DATA, LSIO.GPIO6.IO16
SC_P_ENET1_RGMII_RXD0	CONN.ENET1.RGMII_RXD0, VPU.TSI_S0.SYNC, LSIO.GPIO6.IO18
SC_P_ENET1_RGMII_RXD1	CONN.ENET1.RGMII_RXD1, VPU.TSI_S0.DATA, LSIO.GPIO6.IO19
SC_P_ENET1_RGMII_RXD2	CONN.ENET1.RGMII_RXD2, CONN.ENET1.RMII_RX_ER, VPU.TSI_S0.CLK, LSIO.GPIO6.IO20
SC_P_ENET1_RGMII_RXD3	CONN.ENET1.RGMII_RXD3, DMA.UART3.RX, VPU.TSI_S1.CLK, LSIO.GPIO6.IO21
SC_P_ENET1_RGMII_RX_CTL	CONN.ENET1.RGMII_RX_CTL, VPU.TSI_S0.VID, LSIO.GPIO6.IO17
SC_P_ENET1_RGMII_TXC	CONN.ENET1.RGMII_TXC, CONN.ENET1.RCLK50M_OUT, CONN.ENET1.RCLK50M_IN, LSIO.GPIO6.IO10
SC_P_ENET1_RGMII_TXD0	CONN.ENET1.RGMII_TXD0, LSIO.GPIO6.IO12
SC_P_ENET1_RGMII_TXD1	CONN.ENET1.RGMII_TXD1, LSIO.GPIO6.IO13
SC_P_ENET1_RGMII_TXD2	CONN.ENET1.RGMII_TXD2, DMA.UART3.TX, VPU.TSI_S1.VID, LSIO.GPIO6.IO14
SC_P_ENET1_RGMII_TXD3	CONN.ENET1.RGMII_TXD3, DMA.UART3.RTS_B, VPU.TSI_S1.SYNC, LSIO.GPIO6.IO15
SC_P_ENET1_RGMII_TX_CTL	CONN.ENET1.RGMII_TX_CTL, LSIO.GPIO6.IO11
SC_P_ESAI0_FSR	AUD.ESAI0.FSR, LSIO.GPIO2.IO22
SC_P_ESAI0_FST	AUD.ESAI0.FST, LSIO.GPIO2.IO23
SC_P_ESAI0_SCKR	AUD.ESAI0.SCKR, LSIO.GPIO2.IO24

Pad	Mux Options
SC_P_ESAI0_SCKT	AUD.ESAI0.SCKT, LSIO.GPIO2.IO25
SC_P_ESAI0_TX0	AUD.ESAI0.TX0, LSIO.GPIO2.IO26
SC_P_ESAI0_TX1	AUD.ESAI0.TX1, LSIO.GPIO2.IO27
SC_P_ESAI0_TX2_RX3	AUD.ESAI0.TX2_RX3, LSIO.GPIO2.IO28
SC_P_ESAI0_TX3_RX2	AUD.ESAI0.TX3_RX2, LSIO.GPIO2.IO29
SC_P_ESAI0_TX4_RX1	AUD.ESAI0.TX4_RX1, LSIO.GPIO2.IO30
SC_P_ESAI0_TX5_RX0	AUD.ESAI0.TX5_RX0, LSIO.GPIO2.IO31
SC_P_ESAI1_FSR	AUD.ESAI1.FSR, LSIO.GPIO2.IO04
SC_P_ESAI1_FST	AUD.ESAI1.FST, AUD.SPDIF0.EXT_CLK, LSIO.GPIO2.IO05
SC_P_ESAI1_SCKR	AUD.ESAI1.SCKR, LSIO.GPIO2.IO06
SC_P_ESAI1_SCKT	AUD.ESAI1.SCKT, AUD.SAI2.RXC, AUD.SPDIF0.EXT_CLK, LSIO.GPIO2.IO07
SC_P_ESAI1_TX0	AUD.ESAI1.TX0, AUD.SAI2.RXD, AUD.SPDIF0.RX, LSIO.GPIO2.IO08
SC_P_ESAI1_TX1	AUD.ESAI1.TX1, AUD.SAI2.RXFS, AUD.SPDIF0.TX, LSIO.GPIO2.IO09
SC_P_ESAI1_TX2_RX3	AUD.ESAI1.TX2_RX3, AUD.SPDIF0.RX, LSIO.GPIO2.IO10
SC_P_ESAI1_TX3_RX2	AUD.ESAI1.TX3_RX2, AUD.SPDIF0.TX, LSIO.GPIO2.IO11
SC_P_ESAI1_TX4_RX1	AUD.ESAI1.TX4_RX1, LSIO.GPIO2.IO12
SC_P_ESAI1_TX5_RX0	AUD.ESAI1.TX5_RX0, LSIO.GPIO2.IO13
SC_P_FLEXCAN0_RX	DMA.FLEXCAN0.RX, LSIO.GPIO3.IO29
SC_P_FLEXCAN0_TX	DMA.FLEXCAN0.TX, LSIO.GPIO3.IO30
SC_P_FLEXCAN1_RX	DMA.FLEXCAN1.RX, LSIO.GPIO3.IO31
SC_P_FLEXCAN1_TX	DMA.FLEXCAN1.TX, LSIO.GPIO4.IO00
SC_P_FLEXCAN2_RX	DMA.FLEXCAN2.RX, LSIO.GPIO4.IO01
SC_P_FLEXCAN2_TX	DMA.FLEXCAN2.TX, LSIO.GPIO4.IO02
SC_P_GPT0_CAPTURE	LSIO.GPT0.CAPTURE, DMA.I2C1.SDA, LSIO.KPP0.COL5, LSIO.GPIO0.IO15
SC_P_GPT0_CLK	LSIO.GPT0.CLK, DMA.I2C1.SCL, LSIO.KPP0.COL4, LSIO.GPIO0.IO14
SC_P_GPT0_COMPARE	LSIO.GPT0.COMPARE, LSIO.PWM3.OUT, LSIO.KPP0.COL6, LSIO.GPIO0.IO16
SC_P_GPT1_CAPTURE	LSIO.GPT1.CAPTURE, DMA.I2C2.SDA, LSIO.KPP0.ROW4, LSIO.GPIO0.IO18
SC_P_GPT1_CLK	LSIO.GPT1.CLK, DMA.I2C2.SCL, LSIO.KPP0.COL7, LSIO.GPIO0.IO17
SC_P_GPT1_COMPARE	LSIO.GPT1.COMPARE, LSIO.PWM2.OUT, LSIO.KPP0.ROW5, LSIO.GPIO0.IO19
SC_P_HDMI_TX0_TS_SCL	HDMI_TX0.I2C0.SCL, DMA.I2C0.SCL, LSIO.GPIO2.IO02
SC_P_HDMI_TX0_TS_SDA	HDMI_TX0.I2C0.SDA, DMA.I2C0.SDA, LSIO.GPIO2.IO03
SC_P_LVDS0_GPIO00	LVDS0.GPIO0.IO00, LVDS0.PWM0.OUT, LSIO.GPIO1.IO04
SC_P_LVDS0_GPIO01	LVDS0.GPIO0.IO01, LSIO.GPIO1.IO05

Pad	Mux Options
SC_P_LVDS0_I2C0_SCL	LVDS0.I2C0.SCL, LVDS0.GPIO0.IO02, LSIO.GPIO1.I↔IO06
SC_P_LVDS0_I2C0_SDA	LVDS0.I2C0.SDA, LVDS0.GPIO0.IO03, LSIO.GPIO1.I↔IO07
SC_P_LVDS0_I2C1_SCL	LVDS0.I2C1.SCL, DMA.UART2.TX, LSIO.GPIO1.IO08
SC_P_LVDS0_I2C1_SDA	LVDS0.I2C1.SDA, DMA.UART2.RX, LSIO.GPIO1.IO09
SC_P_LVDS1_GPIO00	LVDS1.GPIO0.IO00, LVDS1.PWM0.OUT, LSIO.GPI↔O1.IO10
SC_P_LVDS1_GPIO01	LVDS1.GPIO0.IO01, LSIO.GPIO1.IO11
SC_P_LVDS1_I2C0_SCL	LVDS1.I2C0.SCL, LVDS1.GPIO0.IO02, LSIO.GPIO1.I↔IO12
SC_P_LVDS1_I2C0_SDA	LVDS1.I2C0.SDA, LVDS1.GPIO0.IO03, LSIO.GPIO1.I↔IO13
SC_P_LVDS1_I2C1_SCL	LVDS1.I2C1.SCL, DMA.UART3.TX, LSIO.GPIO1.IO14
SC_P_LVDS1_I2C1_SDA	LVDS1.I2C1.SDA, DMA.UART3.RX, LSIO.GPIO1.IO15
SC_P_M40_GPIO0_00	M40.GPIO0.IO00, M40.TPM0.CH0, DMA.UART4.RX, LSIO.GPIO0.IO08
SC_P_M40_GPIO0_01	M40.GPIO0.IO01, M40.TPM0.CH1, DMA.UART4.TX, LSIO.GPIO0.IO09
SC_P_M40_I2C0_SCL	M40.I2C0.SCL, M40.UART0.RX, M40.GPIO0.IO02, L↔SIO.GPIO0.IO06
SC_P_M40_I2C0_SDA	M40.I2C0.SDA, M40.UART0.TX, M40.GPIO0.IO03, L↔SIO.GPIO0.IO07
SC_P_M41_GPIO0_00	M41.GPIO0.IO00, M41.TPM0.CH0, DMA.UART3.RX, LSIO.GPIO0.IO12
SC_P_M41_GPIO0_01	M41.GPIO0.IO01, M41.TPM0.CH1, DMA.UART3.TX, LSIO.GPIO0.IO13
SC_P_M41_I2C0_SCL	M41.I2C0.SCL, M41.UART0.RX, M41.GPIO0.IO02, L↔SIO.GPIO0.IO10
SC_P_M41_I2C0_SDA	M41.I2C0.SDA, M41.UART0.TX, M41.GPIO0.IO03, L↔SIO.GPIO0.IO11
SC_P_MCLK_IN0	AUD.ACM.MCLK_IN0, AUD.ESAI0.RX_HF_CLK, AU↔D.ESAI1.RX_HF_CLK, LSIO.GPIO3.IO00
SC_P_MCLK_OUT0	AUD.ACM.MCLK_OUT0, AUD.ESAI0.TX_HF_CLK, A↔UD.ESAI1.TX_HF_CLK, LSIO.GPIO3.IO01
SC_P_MIPI_CSI0_GPIO0_00	MIPI_CSI0.GPIO0.IO00, DMA.I2C0.SCL, MIPI_CSI1.I↔I2C0.SCL, LSIO.GPIO1.IO27
SC_P_MIPI_CSI0_GPIO0_01	MIPI_CSI0.GPIO0.IO01, DMA.I2C0.SDA, MIPI_CSI1.I↔I2C0.SDA, LSIO.GPIO1.IO28
SC_P_MIPI_CSI0_I2C0_SCL	MIPI_CSI0.I2C0.SCL, LSIO.GPIO1.IO25
SC_P_MIPI_CSI0_I2C0_SDA	MIPI_CSI0.I2C0.SDA, LSIO.GPIO1.IO26
SC_P_MIPI_CSI0_MCLK_OUT	MIPI_CSI0.ACM.MCLK_OUT, LSIO.GPIO1.IO24
SC_P_MIPI_CSI1_GPIO0_00	MIPI_CSI1.GPIO0.IO00, DMA.UART4.RX, LSIO.GPI↔O1.IO30
SC_P_MIPI_CSI1_GPIO0_01	MIPI_CSI1.GPIO0.IO01, DMA.UART4.TX, LSIO.GPI↔O1.IO31
SC_P_MIPI_CSI1_I2C0_SCL	MIPI_CSI1.I2C0.SCL, LSIO.GPIO2.IO00
SC_P_MIPI_CSI1_I2C0_SDA	MIPI_CSI1.I2C0.SDA, LSIO.GPIO2.IO01
SC_P_MIPI_CSI1_MCLK_OUT	MIPI_CSI1.ACM.MCLK_OUT, LSIO.GPIO1.IO29

Pad	Mux Options
SC_P_MIPI_DSI0_GPIO0_00	MIPI_DSI0.GPIO0.IO00, MIPI_DSI0.PWM0.OUT, LSI↔O.GPIO1.IO18
SC_P_MIPI_DSI0_GPIO0_01	MIPI_DSI0.GPIO0.IO01, LSIO.GPIO1.IO19
SC_P_MIPI_DSI0_I2C0_SCL	MIPI_DSI0.I2C0.SCL, LSIO.GPIO1.IO16
SC_P_MIPI_DSI0_I2C0_SDA	MIPI_DSI0.I2C0.SDA, LSIO.GPIO1.IO17
SC_P_MIPI_DSI1_GPIO0_00	MIPI_DSI1.GPIO0.IO00, MIPI_DSI1.PWM0.OUT, LSI↔O.GPIO1.IO22
SC_P_MIPI_DSI1_GPIO0_01	MIPI_DSI1.GPIO0.IO01, LSIO.GPIO1.IO23
SC_P_MIPI_DSI1_I2C0_SCL	MIPI_DSI1.I2C0.SCL, LSIO.GPIO1.IO20
SC_P_MIPI_DSI1_I2C0_SDA	MIPI_DSI1.I2C0.SDA, LSIO.GPIO1.IO21
SC_P_MLB_CLK	CONN.MLB.CLK, AUD.SAI3.RXFS, LSIO.GPIO3.IO27
SC_P_MLB_DATA	CONN.MLB.DATA, AUD.SAI3.RXD, LSIO.GPIO3.IO28
SC_P_MLB_SIG	CONN.MLB.SIG, AUD.SAI3.RXC, LSIO.GPIO3.IO26
SC_P_PCIE_CTRL0_CLKREQ_B	HSIO.PCIE0.CLKREQ_B, LSIO.GPIO4.IO27
SC_P_PCIE_CTRL0_PERST_B	HSIO.PCIE0.PERST_B, LSIO.GPIO4.IO29
SC_P_PCIE_CTRL0_WAKE_B	HSIO.PCIE0.WAKE_B, LSIO.GPIO4.IO28
SC_P_PCIE_CTRL1_CLKREQ_B	HSIO.PCIE1.CLKREQ_B, DMA.I2C1.SDA, CONN.USB↔B_OTG2.OC, LSIO.GPIO4.IO30
SC_P_PCIE_CTRL1_PERST_B	HSIO.PCIE1.PERST_B, DMA.I2C1.SCL, CONN.USB↔_OTG1.PWR, LSIO.GPIO5.IO00
SC_P_PCIE_CTRL1_WAKE_B	HSIO.PCIE1.WAKE_B, DMA.I2C1.SCL, CONN.USB↔_OTG2.PWR, LSIO.GPIO4.IO31
SC_P_PMIC_EARLY_WARNING	SCU.PMIC_EARLY_WARNING
SC_P_PMIC_I2C_SCL	SCU.PMIC_I2C.SCL
SC_P_PMIC_I2C_SDA	SCU.PMIC_I2C.SDA
SC_P_PMIC_INT_B	SCU.DSC.PMIC_INT_B
SC_P_QSPI0A_DATA0	LSIO.QSPI0A.DATA0
SC_P_QSPI0A_DATA1	LSIO.QSPI0A.DATA1
SC_P_QSPI0A_DATA2	LSIO.QSPI0A.DATA2
SC_P_QSPI0A_DATA3	LSIO.QSPI0A.DATA3
SC_P_QSPI0A_DQS	LSIO.QSPI0A.DQS
SC_P_QSPI0A_SCLK	LSIO.QSPI0A.SCLK
SC_P_QSPI0A_SS0_B	LSIO.QSPI0A.SS0_B
SC_P_QSPI0A_SS1_B	LSIO.QSPI0A.SS1_B, LSIO.QSPI0A.SCLK2
SC_P_QSPI0B_DATA0	LSIO.QSPI0B.DATA0
SC_P_QSPI0B_DATA1	LSIO.QSPI0B.DATA1
SC_P_QSPI0B_DATA2	LSIO.QSPI0B.DATA2
SC_P_QSPI0B_DATA3	LSIO.QSPI0B.DATA3
SC_P_QSPI0B_DQS	LSIO.QSPI0B.DQS
SC_P_QSPI0B_SCLK	LSIO.QSPI0B.SCLK
SC_P_QSPI0B_SS0_B	LSIO.QSPI0B.SS0_B
SC_P_QSPI0B_SS1_B	LSIO.QSPI0B.SS1_B, LSIO.QSPI0B.SCLK2
SC_P_QSPI1A_DATA0	LSIO.QSPI1A.DATA0, LSIO.GPIO4.IO26
SC_P_QSPI1A_DATA1	LSIO.QSPI1A.DATA1, DMA.I2C1.SDA, CONN.USB↔_OTG2.OC, LSIO.GPIO4.IO25
SC_P_QSPI1A_DATA2	LSIO.QSPI1A.DATA2, DMA.I2C1.SCL, CONN.USB↔_OTG2.PWR, LSIO.GPIO4.IO24

Pad	Mux Options
SC_P_QSPI1A_DATA3	LSIO.QSPI1A.DATA3, DMA.I2C1.SDA, CONN.USB_↔ OTG1.OC, LSIO.GPIO4.IO23
SC_P_QSPI1A_DQS	LSIO.QSPI1A.DQS, LSIO.GPIO4.IO22
SC_P_QSPI1A_SCLK	LSIO.QSPI1A.SCLK, LSIO.GPIO4.IO21
SC_P_QSPI1A_SS0_B	LSIO.QSPI1A.SS0_B, LSIO.GPIO4.IO19
SC_P_QSPI1A_SS1_B	LSIO.QSPI1A.SS1_B, LSIO.QSPI1A.SCLK2, LSIO.G↔ PIO4.IO20
SC_P_SAI1_RXC	AUD.SAI1.RXC, AUD.SAI0.TXD, LSIO.GPIO3.IO12
SC_P_SAI1_RXD	AUD.SAI1.RXD, AUD.SAI0.TXFS, LSIO.GPIO3.IO13
SC_P_SAI1_RXFS	AUD.SAI1.RXFS, AUD.SAI0.RXD, LSIO.GPIO3.IO14
SC_P_SAI1_TXC	AUD.SAI1.TXC, AUD.SAI0.TXC, LSIO.GPIO3.IO15
SC_P_SAI1_TXD	AUD.SAI1.TXD, AUD.SAI1.RXC, LSIO.GPIO3.IO16
SC_P_SAI1_TXFS	AUD.SAI1.TXFS, AUD.SAI1.RXFS, LSIO.GPIO3.IO17
SC_P_SCU_BOOT_MODE0	SCU.DSC.BOOT_MODE0
SC_P_SCU_BOOT_MODE1	SCU.DSC.BOOT_MODE1
SC_P_SCU_BOOT_MODE2	SCU.DSC.BOOT_MODE2
SC_P_SCU_BOOT_MODE3	SCU.DSC.BOOT_MODE3
SC_P_SCU_BOOT_MODE4	SCU.DSC.BOOT_MODE4, SCU.PMIC_I2C.SCL
SC_P_SCU_BOOT_MODE5	SCU.DSC.BOOT_MODE5, SCU.PMIC_I2C.SDA
SC_P_SCU_GPIO0_00	SCU.GPIO0.IO00, SCU.UART0.RX, LSIO.GPIO0.IO28
SC_P_SCU_GPIO0_01	SCU.GPIO0.IO01, SCU.UART0.TX, LSIO.GPIO0.IO29
SC_P_SCU_GPIO0_02	SCU.GPIO0.IO02, SCU.GPIO0.IOXX_PMIC_GPU0_↔ ON, LSIO.GPIO0.IO30
SC_P_SCU_GPIO0_03	SCU.GPIO0.IO03, SCU.GPIO0.IOXX_PMIC_GPU1_↔ ON, LSIO.GPIO0.IO31
SC_P_SCU_GPIO0_04	SCU.GPIO0.IO04, SCU.GPIO0.IOXX_PMIC_A72_ON, LSIO.GPIO1.IO00
SC_P_SCU_GPIO0_05	SCU.GPIO0.IO05, SCU.GPIO0.IOXX_PMIC_A53_ON, LSIO.GPIO1.IO01
SC_P_SCU_GPIO0_06	SCU.GPIO0.IO06, SCU.TPM0.CH0, LSIO.GPIO1.IO02
SC_P_SCU_GPIO0_07	SCU.GPIO0.IO07, SCU.TPM0.CH1, SCU.DSC.RTC_↔ CLOCK_OUTPUT_32K, LSIO.GPIO1.IO03
SC_P_SCU_PMIC_MEMC_ON	SCU.GPIO0.IOXX_PMIC_MEMC_ON
SC_P_SCU_WDOG_OUT	SCU.WDOG0.WDOG_OUT
SC_P_SIM0_CLK	DMA.SIM0.CLK, LSIO.GPIO0.IO00
SC_P_SIM0_GPIO0_00	DMA.SIM0.POWER_EN, LSIO.GPIO0.IO05
SC_P_SIM0_IO	DMA.SIM0.IO, LSIO.GPIO0.IO02
SC_P_SIM0_PD	DMA.SIM0.PD, DMA.I2C3.SCL, LSIO.GPIO0.IO03
SC_P_SIM0_POWER_EN	DMA.SIM0.POWER_EN, DMA.I2C3.SDA, LSIO.GPI↔ O0.IO04
SC_P_SIM0_RST	DMA.SIM0.RST, LSIO.GPIO0.IO01
SC_P_SPDIF0_EXT_CLK	AUD.SPDIF0.EXT_CLK, DMA.DMA0.REQ_IN0, LSI↔ O.GPIO2.IO16
SC_P_SPDIF0_RX	AUD.SPDIF0.RX, AUD.MQS.R, AUD.ACM.MCLK_IN1, LSIO.GPIO2.IO14
SC_P_SPDIF0_TX	AUD.SPDIF0.TX, AUD.MQS.L, AUD.ACM.MCLK_OU↔ T1, LSIO.GPIO2.IO15
SC_P_SPI0_CS0	DMA.SPI0.CS0, AUD.SAI0.RXFS, LSIO.GPIO3.IO05
SC_P_SPI0_CS1	DMA.SPI0.CS1, AUD.SAI0.TXC, LSIO.GPIO3.IO06

Pad	Mux Options
SC_P_SPI0_SCK	DMA.SPI0.SCK, AUD.SAI0.RXC, LSIO.GPIO3.IO02
SC_P_SPI0_SDI	DMA.SPI0.SDI, AUD.SAI0.RXD, LSIO.GPIO3.IO04
SC_P_SPI0_SDO	DMA.SPI0.SDO, AUD.SAI0.TXD, LSIO.GPIO3.IO03
SC_P_SPI2_CS0	DMA.SPI2.CS0, LSIO.GPIO3.IO10
SC_P_SPI2_CS1	DMA.SPI2.CS1, AUD.SAI0.TXFS, LSIO.GPIO3.IO11
SC_P_SPI2_SCK	DMA.SPI2.SCK, LSIO.GPIO3.IO07
SC_P_SPI2_SDI	DMA.SPI2.SDI, LSIO.GPIO3.IO09
SC_P_SPI2_SDO	DMA.SPI2.SDO, LSIO.GPIO3.IO08
SC_P_SPI3_CS0	DMA.SPI3.CS0, DMA.FTM.CH2, LSIO.GPIO2.IO20
SC_P_SPI3_CS1	DMA.SPI3.CS1, LSIO.GPIO2.IO21
SC_P_SPI3_SCK	DMA.SPI3.SCK, LSIO.GPIO2.IO17
SC_P_SPI3_SDI	DMA.SPI3.SDI, DMA.FTM.CH1, LSIO.GPIO2.IO19
SC_P_SPI3_SDO	DMA.SPI3.SDO, DMA.FTM.CH0, LSIO.GPIO2.IO18
SC_P_UART0_CTS_B	DMA.UART0.CTS_B, LSIO.PWM1.OUT, DMA.UART0.TX, LSIO.GPIO0.IO23
SC_P_UART0_RTS_B	DMA.UART0.RTS_B, LSIO.PWM0.OUT, DMA.UART0.RX, LSIO.GPIO0.IO22
SC_P_UART0_RX	DMA.UART0.RX, SCU.UART0.RX, LSIO.GPIO0.IO20
SC_P_UART0_TX	DMA.UART0.TX, SCU.UART0.TX, LSIO.GPIO0.IO21
SC_P_UART1_CTS_B	DMA.UART1.CTS_B, DMA.SPI3.CS0, DMA.UART1.RTS_B, LSIO.GPIO0.IO27
SC_P_UART1_RTS_B	DMA.UART1.RTS_B, DMA.SPI3.SDI, DMA.UART1.CTS_B, LSIO.GPIO0.IO26
SC_P_UART1_RX	DMA.UART1.RX, DMA.SPI3.SDO, LSIO.GPIO0.IO25
SC_P_UART1_TX	DMA.UART1.TX, DMA.SPI3.SCK, LSIO.GPIO0.IO24
SC_P_USB_HSIC0_DATA	CONN.USB_HSIC0.DATA, DMA.I2C1.SDA, LSIO.GPIO5.IO01
SC_P_USB_HSIC0_STROBE	CONN.USB_HSIC0.STROBE, DMA.I2C1.SCL, LSIO.GPIO5.IO02
SC_P_USB_SS3_TC0	DMA.I2C1.SCL, CONN.USB_OTG1.PWR, LSIO.GPIO4.IO03
SC_P_USB_SS3_TC1	DMA.I2C1.SCL, CONN.USB_OTG2.PWR, LSIO.GPIO4.IO04
SC_P_USB_SS3_TC2	DMA.I2C1.SDA, CONN.USB_OTG1.OC, LSIO.GPIO4.IO05
SC_P_USB_SS3_TC3	DMA.I2C1.SDA, CONN.USB_OTG2.OC, LSIO.GPIO4.IO06
SC_P_USDHC1_CLK	CONN.USDHC1.CLK, AUD.MQS.R
SC_P_USDHC1_CMD	CONN.USDHC1.CMD, AUD.MQS.L, LSIO.GPIO5.IO14
SC_P_USDHC1_DATA0	CONN.USDHC1.DATA0, CONN.NAND.RE_N, LSIO.GPIO5.IO15
SC_P_USDHC1_DATA1	CONN.USDHC1.DATA1, CONN.NAND.RE_P, LSIO.GPIO5.IO16
SC_P_USDHC1_DATA2	CONN.USDHC1.DATA2, CONN.NAND.DQS_N, LSIO.GPIO5.IO17
SC_P_USDHC1_DATA3	CONN.USDHC1.DATA3, CONN.NAND.DQS_P, LSIO.GPIO5.IO18
SC_P_USDHC1_DATA4	CONN.USDHC1.DATA4, CONN.NAND.CE0_B, AUD.MQS.R, LSIO.GPIO5.IO19

Pad	Mux Options
SC_P_USDHC1_DATA5	CONN.USDHC1.DATA5, CONN.NAND.RE_B, AUD.↔ MQS.L, LSIO.GPIO5.IO20
SC_P_USDHC1_DATA6	CONN.USDHC1.DATA6, CONN.NAND.WE_B, CON↔ N.USDHC1.WP, LSIO.GPIO5.IO21
SC_P_USDHC1_DATA7	CONN.USDHC1.DATA7, CONN.NAND.ALE, CONN.↔ USDHC1.CD_B, LSIO.GPIO5.IO22
SC_P_USDHC1_RESET_B	CONN.USDHC1.RESET_B, LSIO.GPIO4.IO07
SC_P_USDHC1_STROBE	CONN.USDHC1.STROBE, CONN.NAND.CE1_B, CO↔ NN.USDHC1.RESET_B, LSIO.GPIO5.IO23
SC_P_USDHC1_VSELECT	CONN.USDHC1.VSELECT, LSIO.GPIO4.IO08
SC_P_USDHC2_CD_B	CONN.USDHC2.CD_B, LSIO.GPIO4.IO12
SC_P_USDHC2_CLK	CONN.USDHC2.CLK, AUD.MQS.R, LSIO.GPIO5.IO24
SC_P_USDHC2_CMD	CONN.USDHC2.CMD, AUD.MQS.L, LSIO.GPIO5.IO25
SC_P_USDHC2_DATA0	CONN.USDHC2.DATA0, DMA.UART4.RX, LSIO.GPI↔ O5.IO26
SC_P_USDHC2_DATA1	CONN.USDHC2.DATA1, DMA.UART4.TX, LSIO.GPI↔ O5.IO27
SC_P_USDHC2_DATA2	CONN.USDHC2.DATA2, DMA.UART4.CTS_B, LSIO.↔ GPIO5.IO28
SC_P_USDHC2_DATA3	CONN.USDHC2.DATA3, DMA.UART4.RTS_B, LSIO.↔ GPIO5.IO29
SC_P_USDHC2_RESET_B	CONN.USDHC2.RESET_B, LSIO.GPIO4.IO09
SC_P_USDHC2_VSELECT	CONN.USDHC2.VSELECT, LSIO.GPIO4.IO10
SC_P_USDHC2_WP	CONN.USDHC2.WP, LSIO.GPIO4.IO11

Chapter 8

Module Index

8.1 Modules

Here is a list of all modules:

(SVC) Interrupt Service	45
(SVC) Miscellaneous Service	50
(SVC) Pad Service	72
(SVC) Power Management Service	90
(SVC) Resource Management Service	112
(SVC) Timer Service	138

Chapter 9

File Index

9.1 File List

Here is a list of all documented files with brief descriptions:

platform/main/ ipc.h	
Header file for the IPC implementation	151
platform/main/ types.h	
Header file containing types used across multiple service APIs	154
platform/svc/irq/ api.h	
Header file containing the public API for the System Controller (SC) Interrupt (IRQ) function	170
platform/svc/misc/ api.h	
Header file containing the public API for the System Controller (SC) Miscellaneous (MISC) function	173
platform/svc/pad/ api.h	
Header file containing the public API for the System Controller (SC) Pad Control (PAD) function	176
platform/svc/pm/ api.h	
Header file containing the public API for the System Controller (SC) Power Management (PM) function	179
platform/svc/rm/ api.h	
Header file containing the public API for the System Controller (SC) Resource Management (RM) function	184
platform/svc/timer/ api.h	
Header file containing the public API for the System Controller (SC) Timer function	187

Chapter 10

Module Documentation

10.1 (SVC) Interrupt Service

Module for the Interrupt (IRQ) service.

Macros

- `#define SC_IRQ_NUM_GROUP 5U`
Number of groups.

Typedefs

- `typedef uint8_t sc_irq_group_t`
This type is used to declare an interrupt group.
- `typedef uint8_t sc_irq_temp_t`
This type is used to declare a bit mask of temp interrupts.
- `typedef uint8_t sc_irq_wdog_t`
This type is used to declare a bit mask of watchdog interrupts.
- `typedef uint8_t sc_irq_rtc_t`
This type is used to declare a bit mask of RTC interrupts.
- `typedef uint8_t sc_irq_wake_t`
This type is used to declare a bit mask of wakeup interrupts.

Functions

- `sc_err_t sc_irq_enable` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_irq_group_t group`, `uint32_t mask`, `sc_bool_t enable`)
This function enables/disables interrupts.
- `sc_err_t sc_irq_status` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_irq_group_t group`, `uint32_t *status`)
This function returns the current interrupt status (regardless if masked).

Defines for `sc_irq_group_t`

- `#define SC_IRQ_GROUP_TEMP 0U`
Temp interrupts.
- `#define SC_IRQ_GROUP_WDOG 1U`
Watchdog interrupts.
- `#define SC_IRQ_GROUP_RTC 2U`
RTC interrupts.
- `#define SC_IRQ_GROUP_WAKE 3U`
Wakeup interrupts.
- `#define SC_IRQ_GROUP_SYSCTR 4U`
System counter interrupts.

Defines for `sc_irq_temp_t`

- `#define SC_IRQ_TEMP_HIGH (1UL << 0U)`
Temp alarm interrupt.
- `#define SC_IRQ_TEMP_CPU0_HIGH (1UL << 1U)`
CPU0 temp alarm interrupt.
- `#define SC_IRQ_TEMP_CPU1_HIGH (1UL << 2U)`
CPU1 temp alarm interrupt.
- `#define SC_IRQ_TEMP_GPU0_HIGH (1UL << 3U)`
GPU0 temp alarm interrupt.
- `#define SC_IRQ_TEMP_GPU1_HIGH (1UL << 4U)`
GPU1 temp alarm interrupt.
- `#define SC_IRQ_TEMP_DRC0_HIGH (1UL << 5U)`
DRC0 temp alarm interrupt.
- `#define SC_IRQ_TEMP_DRC1_HIGH (1UL << 6U)`
DRC1 temp alarm interrupt.
- `#define SC_IRQ_TEMP_VPU_HIGH (1UL << 7U)`
DRC1 temp alarm interrupt.
- `#define SC_IRQ_TEMP_PMIC0_HIGH (1UL << 8U)`
PMIC0 temp alarm interrupt.
- `#define SC_IRQ_TEMP_PMIC1_HIGH (1UL << 9U)`
PMIC1 temp alarm interrupt.
- `#define SC_IRQ_TEMP_LOW (1UL << 10U)`
Temp alarm interrupt.
- `#define SC_IRQ_TEMP_CPU0_LOW (1UL << 11U)`
CPU0 temp alarm interrupt.
- `#define SC_IRQ_TEMP_CPU1_LOW (1UL << 12U)`
CPU1 temp alarm interrupt.
- `#define SC_IRQ_TEMP_GPU0_LOW (1UL << 13U)`
GPU0 temp alarm interrupt.
- `#define SC_IRQ_TEMP_GPU1_LOW (1UL << 14U)`
GPU1 temp alarm interrupt.
- `#define SC_IRQ_TEMP_DRC0_LOW (1UL << 15U)`
DRC0 temp alarm interrupt.

- #define `SC_IRQ_TEMP_DRC1_LOW` (1UL << 16U)
DRC1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_VPU_LOW` (1UL << 17U)
DRC1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_PMIC0_LOW` (1UL << 18U)
PMIC0 temp alarm interrupt.
- #define `SC_IRQ_TEMP_PMIC1_LOW` (1UL << 19U)
PMIC1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_PMIC2_HIGH` (1UL << 20U)
PMIC2 temp alarm interrupt.
- #define `SC_IRQ_TEMP_PMIC2_LOW` (1UL << 21U)
PMIC2 temp alarm interrupt.

Defines for `sc_irq_wdog_t`

- #define `SC_IRQ_WDOG` (1U << 0U)
Watchdog interrupt.

Defines for `sc_irq_rtc_t`

- #define `SC_IRQ_RTC` (1U << 0U)
RTC interrupt.

Defines for `sc_irq_wake_t`

- #define `SC_IRQ_BUTTON` (1U << 0U)
Button interrupt.
- #define `SC_IRQ_PAD` (1U << 1U)
Pad wakeup.

Defines for `sc_irq_sysctr_t`

- #define `SC_IRQ_SYSCTR` (1U << 0U)
SYSCTR interrupt.

10.1.1 Detailed Description

Module for the Interrupt (IRQ) service.

10.1.2 Function Documentation

10.1.2.1 sc_irq_enable()

```
sc_err_t sc_irq_enable (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_irq_group_t group,
    uint32_t mask,
    sc_bool_t enable )
```

This function enables/disables interrupts.

If pending interrupts are unmasked, an interrupt will be triggered.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	MU channel
in	<i>group</i>	group the interrupts are in
in	<i>mask</i>	mask of interrupts to affect
in	<i>enable</i>	state to change interrupts to

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if group invalid

10.1.2.2 sc_irq_status()

```
sc_err_t sc_irq_status (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_irq_group_t group,
    uint32_t * status )
```

This function returns the current interrupt status (regardless if masked).

Automatically clears pending interrupts.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	MU channel
in	<i>group</i>	groups the interrupts are in
in	<i>status</i>	status of interrupts

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if group invalid

The returned *status* may show interrupts pending that are currently masked.

10.2 (SVC) Miscellaneous Service

Module for the Miscellaneous (MISC) service.

Macros

- #define `SC_MISC_DMA_GRP_MAX` 31U
Max DMA channel priority group.

Typedefs

- typedef `uint8_t sc_misc_dma_group_t`
This type is used to store a DMA channel priority group.
- typedef `uint8_t sc_misc_boot_status_t`
This type is used report boot status.
- typedef `uint8_t sc_misc_seco_auth_cmd_t`
This type is used to issue SECO authenticate commands.
- typedef `uint8_t sc_misc_temp_t`
This type is used report boot status.
- typedef `uint8_t sc_misc_bt_t`
This type is used report the boot type.

Defines for type widths

- #define `SC_MISC_DMA_GRP_W` 5U
Width of `sc_misc_dma_group_t`.

Defines for `sc_misc_boot_status_t`

- #define `SC_MISC_BOOT_STATUS_SUCCESS` 0U
Success.
- #define `SC_MISC_BOOT_STATUS_SECURITY` 1U
Security violation.

Defines for `sc_misc_temp_t`

- #define `SC_MISC_TEMP` 0U
Temp sensor.
- #define `SC_MISC_TEMP_HIGH` 1U
Temp high alarm.
- #define `SC_MISC_TEMP_LOW` 2U
Temp low alarm.

Defines for `sc_misc_seco_auth_cmd_t`

- `#define SC_MISC_AUTH_CONTAINER 0U`
Authenticate container.
- `#define SC_MISC_VERIFY_IMAGE 1U`
Verify image.
- `#define SC_MISC_REL_CONTAINER 2U`
Release container.
- `#define SC_MISC_SECO_AUTH_SECO_FW 3U`
SECO Firmware.
- `#define SC_MISC_SECO_AUTH_HDMI_TX_FW 4U`
HDMI TX Firmware.
- `#define SC_MISC_SECO_AUTH_HDMI_RX_FW 5U`
HDMI RX Firmware.

Defines for `sc_misc_bt_t`

- `#define SC_MISC_BT_PRIMARY 0U`
- `#define SC_MISC_BT_SECONDARY 1U`
- `#define SC_MISC_BT_RECOVERY 2U`
- `#define SC_MISC_BT_MANUFACTURE 3U`
- `#define SC_MISC_BT_SERIAL 4U`

Control Functions

- `sc_err_t sc_misc_set_control` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_ctrl_t` ctrl, `uint32_t` val)
This function sets a miscellaneous control value.
- `sc_err_t sc_misc_get_control` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_ctrl_t` ctrl, `uint32_t` *val)
This function gets a miscellaneous control value.

DMA Functions

- `sc_err_t sc_misc_set_max_dma_group` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_misc_dma_group_t` max)
This function configures the max DMA channel priority group for a partition.
- `sc_err_t sc_misc_set_dma_group` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_misc_dma_group_t` group)
This function configures the priority group for a DMA channel.

Security Functions

- `sc_err_t sc_misc_seco_image_load` (`sc_ipc_t` ipc, `sc_faddr_t` addr_src, `sc_faddr_t` addr_dst, `uint32_t` len, `sc_bool_t` fw)
This function loads a SECO image.
- `sc_err_t sc_misc_seco_authenticate` (`sc_ipc_t` ipc, `sc_misc_seco_auth_cmd_t` cmd, `sc_faddr_t` addr)
This function is used to authenticate a SECO image or command.
- `sc_err_t sc_misc_seco_fuse_write` (`sc_ipc_t` ipc, `sc_faddr_t` addr)
This function securely writes a group of fuse words.
- `sc_err_t sc_misc_seco_enable_debug` (`sc_ipc_t` ipc, `sc_faddr_t` addr)
This function securely enables debug.
- `sc_err_t sc_misc_seco_forward_lifecycle` (`sc_ipc_t` ipc, `uint32_t` change)
This function updates the lifecycle of the device.
- `sc_err_t sc_misc_seco_return_lifecycle` (`sc_ipc_t` ipc, `sc_faddr_t` addr)
This function updates the lifecycle to one of the return lifecycles.
- `void sc_misc_seco_build_info` (`sc_ipc_t` ipc, `uint32_t` *version, `uint32_t` *commit)
This function is used to return the SECO FW build info.
- `sc_err_t sc_misc_seco_chip_info` (`sc_ipc_t` ipc, `uint16_t` *lc, `uint16_t` *monotonic, `uint32_t` *uid_l, `uint32_t` *uid_h)
This function is used to return SECO chip info.
- `sc_err_t sc_misc_seco_attest_mode` (`sc_ipc_t` ipc, `uint32_t` mode)
This function is used to set the attestation mode.
- `sc_err_t sc_misc_seco_attest` (`sc_ipc_t` ipc, `uint64_t` nonce)
This function is used to request atestation.
- `sc_err_t sc_misc_seco_get_attest_pkey` (`sc_ipc_t` ipc, `sc_faddr_t` addr)
This function is used to retrieve the attestation public key.
- `sc_err_t sc_misc_seco_get_attest_sign` (`sc_ipc_t` ipc, `sc_faddr_t` addr)
This function is used to retrieve attestation signature and parameters.
- `sc_err_t sc_misc_seco_attest_verify` (`sc_ipc_t` ipc, `sc_faddr_t` addr)
This function is used to verify attestation.
- `sc_err_t sc_misc_seco_commit` (`sc_ipc_t` ipc, `uint32_t` *info)
This function is used to commit into the fuses any new SRK revocation and FW version information that have been found in the primary and secondary containers.

Debug Functions

- `void sc_misc_debug_out` (`sc_ipc_t` ipc, `uint8_t` ch)
This function is used output a debug character from the SCU UART.
- `sc_err_t sc_misc_waveform_capture` (`sc_ipc_t` ipc, `sc_bool_t` enable)
This function starts/stops emulation waveform capture.
- `void sc_misc_build_info` (`sc_ipc_t` ipc, `uint32_t` *build, `uint32_t` *commit)
This function is used to return the SCFW build info.
- `void sc_misc_unique_id` (`sc_ipc_t` ipc, `uint32_t` *id_l, `uint32_t` *id_h)
This function is used to return the device's unique ID.

Other Functions

- `sc_err_t sc_misc_set_ari` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_rsrc_t resource_mst`, `uint16_t ari`, `sc_bool_t enable`)
This function configures the ARI match value for PCIe/SATA resources.
- `void sc_misc_boot_status` (`sc_ipc_t ipc`, `sc_misc_boot_status_t status`)
This function reports boot status.
- `sc_err_t sc_misc_boot_done` (`sc_ipc_t ipc`, `sc_rsrc_t cpu`)
This function tells the SCFW that a CPU is done booting.
- `sc_err_t sc_misc_otf_fuse_read` (`sc_ipc_t ipc`, `uint32_t word`, `uint32_t *val`)
This function reads a given fuse word index.
- `sc_err_t sc_misc_otf_fuse_write` (`sc_ipc_t ipc`, `uint32_t word`, `uint32_t val`)
This function writes a given fuse word index.
- `sc_err_t sc_misc_set_temp` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_misc_temp_t temp`, `int16_t celsius`, `int8_t tenths`)
This function sets a temp sensor alarm.
- `sc_err_t sc_misc_get_temp` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_misc_temp_t temp`, `int16_t *celsius`, `int8_t *tenths`)
This function gets a temp sensor value.
- `void sc_misc_get_boot_dev` (`sc_ipc_t ipc`, `sc_rsrc_t *dev`)
This function returns the boot device.
- `sc_err_t sc_misc_get_boot_type` (`sc_ipc_t ipc`, `sc_misc_bt_t *type`)
This function returns the boot type.
- `void sc_misc_get_button_status` (`sc_ipc_t ipc`, `sc_bool_t *status`)
This function returns the current status of the ON/OFF button.
- `sc_err_t sc_misc_rompatch_checksum` (`sc_ipc_t ipc`, `uint32_t *checksum`)
This function returns the ROM patch checksum.

10.2.1 Detailed Description

Module for the Miscellaneous (MISC) service.

10.2.2 Function Documentation

10.2.2.1 `sc_misc_set_control()`

```
sc_err_t sc_misc_set_control (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_ctrl_t ctrl,
    uint32_t val )
```

This function sets a miscellaneous control value.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource the control is associated with
in	<i>ctrl</i>	control to change
in	<i>val</i>	value to apply to the control

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner

Refer to the [Control List](#) for valid control values.

10.2.2.2 sc_misc_get_control()

```
sc_err_t sc_misc_get_control (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_ctrl_t ctrl,
    uint32_t * val )
```

This function gets a miscellaneous control value.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource the control is associated with
in	<i>ctrl</i>	control to get
out	<i>val</i>	pointer to return the control value

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner

Refer to the [Control List](#) for valid control values.

10.2.2.3 `sc_misc_set_max_dma_group()`

```
sc_err_t sc_misc_set_max_dma_group (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_misc_dma_group_t max )
```

This function configures the max DMA channel priority group for a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to assign <i>max</i>
in	<i>max</i>	max priority group (0-31)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the parent of the affected partition

Valid *max* range is 0-31 with 0 being the lowest and 31 the highest. Default is the max priority group for the parent partition of *pt*.

10.2.2.4 `sc_misc_set_dma_group()`

```
sc_err_t sc_misc_set_dma_group (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_misc_dma_group_t group )
```

This function configures the priority group for a DMA channel.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	DMA channel resource
in	<i>group</i>	priority group (0-31)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the owner or parent of the owner of the DMA channel

Valid *group* range is 0-31 with 0 being the lowest and 31 the highest. The max value of *group* is limited by the partition max set using `sc_misc_set_max_dma_group()`.

10.2.2.5 `sc_misc_seco_image_load()`

```
sc_err_t sc_misc_seco_image_load (
    sc_ipc_t ipc,
    sc_faddr_t addr_src,
    sc_faddr_t addr_dst,
    uint32_t len,
    sc_bool_t fw )
```

This function loads a SECO image.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>addr_src</i>	address of image source
in	<i>addr_dst</i>	address of image destination
in	<i>len</i>	length of image to load
in	<i>fw</i>	SC_TRUE = firmware load

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if word fuse index param out of range or invalid
- SC_ERR_UNAVAILABLE if SECO not available

This is used to load images via the SECO. Examples include SECO Firmware and IVT/CSF data used for authentication. These are usually loaded into SECO TCM. *addr_src* is in secure memory.

See the Security Reference Manual (SRM) for more info.

10.2.2.6 `sc_misc_seco_authenticate()`

```
sc_err_t sc_misc_seco_authenticate (
    sc_ipc_t ipc,
    sc_misc_seco_auth_cmd_t cmd,
    sc_faddr_t addr )
```

This function is used to authenticate a SECO image or command.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>cmd</i>	authenticate command
in	<i>addr</i>	address of/or metadata

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if word fuse index param out of range or invalid
- SC_ERR_UNAVAILABLE if SECO not available

This is used to authenticate a SECO image or issue a security command. *addr* often points to an container. It is also just data (or even unused) for some commands.

See the Security Reference Manual (SRM) for more info.

10.2.2.7 `sc_misc_seco_fuse_write()`

```
sc_err_t sc_misc_seco_fuse_write (
    sc_ipc_t ipc,
    sc_faddr_t addr )
```

This function securely writes a group of fuse words.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>addr</i>	address of message block

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available

Note *addr* must be a pointer to a signed message block.

See the Security Reference Manual (SRM) for more info.

10.2.2.8 sc_misc_seco_enable_debug()

```
sc_err_t sc_misc_seco_enable_debug (
    sc_ipc_t ipc,
    sc_faddr_t addr )
```

This function securely enables debug.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>addr</i>	address of message block

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available

Note *addr* must be a pointer to a signed message block.

See the Security Reference Manual (SRM) for more info.

10.2.2.9 sc_misc_seco_forward_lifecycle()

```
sc_err_t sc_misc_seco_forward_lifecycle (
    sc_ipc_t ipc,
    uint32_t change )
```

This function updates the lifecycle of the device.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>change</i>	desired lifecycle transistion

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available

This message is used for going from Open to NXP Closed to OEM Closed. Note *change* is NOT the new desired lifecycle. It is a lifecycle transition as documented in the Security Reference Manual (SRM).

If any SECO request fails or only succeeds because the part is in an "OEM open" lifecycle, then a request to transition from "NXP closed" to "OEM closed" will also fail. For example, booting a signed container when the OEM SRK is not fused will succeed, but as it is an abnormal situation, a subsequent request to transition the lifecycle will return an error.

10.2.2.10 sc_misc_seco_return_lifecycle()

```
sc_err_t sc_misc_seco_return_lifecycle (
    sc_ipc_t ipc,
    sc_faddr_t addr )
```

This function updates the lifecycle to one of the return lifecycles.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>addr</i>	address of message block

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available

Note *addr* must be a pointer to a signed message block.

To switch back to NXP states (Full Field Return), message must be signed by NXP SRK. For OEM States (Partial Field Return), must be signed by OEM SRK.

See the Security Reference Manual (SRM) for more info.

10.2.2.11 sc_misc_seco_build_info()

```
void sc_misc_seco_build_info (
    sc_ipc_t ipc,
    uint32_t * version,
    uint32_t * commit )
```

This function is used to return the SECO FW build info.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>version</i>	pointer to return build number
out	<i>commit</i>	pointer to return commit ID (git SHA-1)

10.2.2.12 sc_misc_seco_chip_info()

```
sc_err_t sc_misc_seco_chip_info (
    sc_ipc_t ipc,
    uint16_t * lc,
    uint16_t * monotonic,
    uint32_t * uid_l,
    uint32_t * uid_h )
```

This function is used to return SECO chip info.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>lc</i>	pointer to return lifecycle
out	<i>monotonic</i>	pointer to return monotonic counter
out	<i>uid_l</i>	pointer to return UID (lower 32 bits)
out	<i>uid_h</i>	pointer to return UID (upper 32 bits)

10.2.2.13 sc_misc_seco_attest_mode()

```
sc_err_t sc_misc_seco_attest_mode (
    sc_ipc_t ipc,
    uint32_t mode )
```

This function is used to set the attestation mode.

Only the owner of the SC_R_ATTESTATION resource may make this call.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mode</i>	mode

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if *mode* is invalid
- SC_ERR_NOACCESS if SC_R_ATTESTATION not owned by caller
- SC_ERR_UNAVAILABLE if SECO not available

This is used to set the SECO attestation mode. This can be prover or verifier. See the Security Reference Manual (SRM) for more on the supported modes, mode values, and mode behavior.

10.2.2.14 sc_misc_seco_attest()

```
sc_err_t sc_misc_seco_attest (
    sc_ipc_t ipc,
    uint64_t nonce )
```

This function is used to request attestation.

Only the owner of the SC_R_ATTESTATION resource may make this call.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>nonce</i>	unique value

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_NOACCESS if SC_R_ATTESTATION not owned by caller
- SC_ERR_UNAVAILABLE if SECO not available

This is used to ask SECO to perform an attestation. The result depends on the attestation mode. After this call, the signature can be requested or a verify can be requested.

See the Security Reference Manual (SRM) for more info.

10.2.2.15 sc_misc_seco_get_attest_pkey()

```
sc_err_t sc_misc_seco_get_attest_pkey (
    sc_ipc_t ipc,
    sc_faddr_t addr )
```

This function is used to retrieve the attestation public key.

Mode must be verifier. Only the owner of the SC_R_ATTESTATION resource may make this call.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>addr</i>	address to write response

Result will be written to *addr*. The *addr* parameter must point to an address SECO can access. It must be 64-bit aligned. There should be 96 bytes of space.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if *addr* bad or attestation has not been requested
- SC_ERR_NOACCESS if SC_R_ATTESTATION not owned by caller
- SC_ERR_UNAVAILABLE if SECO not available

See the Security Reference Manual (SRM) for more info.

10.2.2.16 sc_misc_seco_get_attest_sign()

```
sc_err_t sc_misc_seco_get_attest_sign (
    sc_ipc_t ipc,
    sc_faddr_t addr )
```

This function is used to retrieve attestation signature and parameters.

Mode must be provider. Only the owner of the SC_R_ATTESTATION resource may make this call.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>addr</i>	address to write response

Result will be written to *addr*. The *addr* parameter must point to an address SECO can access. It must be 64-bit aligned. There should be 120 bytes of space.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if *addr* bad or attestation has not been requested
- SC_ERR_NOACCESS if SC_R_ATTESTATION not owned by caller
- SC_ERR_UNAVAILABLE if SECO not available

See the Security Reference Manual (SRM) for more info.

10.2.2.17 sc_misc_seco_attest_verify()

```
sc_err_t sc_misc_seco_attest_verify (  
    sc_ipc_t ipc,  
    sc_faddr_t addr )
```

This function is used to verify attestation.

Mode must be verifier. Only the owner of the SC_R_ATTESTATION resource may make this call.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>addr</i>	address of signature

The *addr* parameter must point to an address SECO can access. It must be 64-bit aligned.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if *addr* bad or attestation has not been requested
- SC_ERR_NOACCESS if SC_R_ATTESTATION not owned by caller
- SC_ERR_UNAVAILABLE if SECO not available
- SC_ERR_FAIL if signature doesn't match

See the Security Reference Manual (SRM) for more info.

10.2.2.18 sc_misc_seco_commit()

```
sc_err_t sc_misc_seco_commit (
    sc_ipc_t ipc,
    uint32_t * info )
```

This function is used to commit into the fuses any new SRK revocation and FW version information that have been found in the primary and secondary containers.

Parameters

in	<i>ipc</i>	IPC handle
in, out	<i>info</i>	pointer to information type to be committed The return <i>info</i> will contain what was actually committed.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if *info* is invalid
- SC_ERR_UNAVAILABLE if SECO not available

10.2.2.19 sc_misc_debug_out()

```
void sc_misc_debug_out (
    sc_ipc_t ipc,
    uint8_t ch )
```

This function is used output a debug character from the SCU UART.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>ch</i>	character to output

10.2.2.20 sc_misc_waveform_capture()

```
sc_err_t sc_misc_waveform_capture (
```



```
sc_ipc_t ipc,  
sc_bool_t enable )
```

This function starts/stops emulation waveform capture.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>enable</i>	flag to enable/disable capture

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_UNAVAILABLE if not running on emulation

10.2.2.21 sc_misc_build_info()

```
void sc_misc_build_info (  
    sc_ipc_t ipc,  
    uint32_t * build,  
    uint32_t * commit )
```

This function is used to return the SCFW build info.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>build</i>	pointer to return build number
out	<i>commit</i>	pointer to return commit ID (git SHA-1)

10.2.2.22 sc_misc_unique_id()

```
void sc_misc_unique_id (  
    sc_ipc_t ipc,  
    uint32_t * id_l,  
    uint32_t * id_h )
```

This function is used to return the device's unique ID.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>id↔ _l</i>	pointer to return lower 32-bit of ID [31:0]
out	<i>id↔ _h</i>	pointer to return upper 32-bits of ID [63:32]

10.2.2.23 sc_misc_set_ari()

```

sc_err_t sc_misc_set_ari (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_rsrc_t resource_mst,
    uint16_t ari,
    sc_bool_t enable )

```

This function configures the ARI match value for PCIe/SATA resources.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	match resource
in	<i>resource_mst</i>	PCIe/SATA master to match
in	<i>ari</i>	ARI to match
in	<i>enable</i>	enable match or not

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the owner or parent of the owner of the resource and translation

For PCIe, the ARI is the 16-bit value that includes the bus number, device number, and function number. For SATA, this value includes the FISType and PM_Port.

10.2.2.24 sc_misc_boot_status()

```

void sc_misc_boot_status (
    sc_ipc_t ipc,
    sc_misc_boot_status_t status )

```

This function reports boot status.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>status</i>	boot status This is used by SW partitions to report status of boot. This is normally used to report a boot failure.

10.2.2.25 sc_misc_boot_done()

```
sc_err_t sc_misc_boot_done (
    sc_ipc_t ipc,
    sc_rsrc_t cpu )
```

This function tells the SCFW that a CPU is done booting.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>cpu</i>	CPU that is done booting

This is called by early booting CPUs to report they are done with initialization. After starting early CPUs, the SCFW halts the booting process until they are done. During this time, early CPUs can call the SCFW with lower latency as the SCFW is idle.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the CPU owner

10.2.2.26 sc_misc_otf_fuse_read()

```
sc_err_t sc_misc_otf_fuse_read (
    sc_ipc_t ipc,
    uint32_t word,
    uint32_t * val )
```

This function reads a given fuse word index.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>word</i>	fuse word index
out	<i>val</i>	fuse read value

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if word fuse index param out of range or invalid
- SC_ERR_NOACCESS if read operation failed
- SC_ERR_LOCKED if read operation is locked

10.2.2.27 sc_misc_otp_fuse_write()

```
sc_err_t sc_misc_otp_fuse_write (
    sc_ipc_t ipc,
    uint32_t word,
    uint32_t val )
```

This function writes a given fuse word index.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>word</i>	fuse word index
in	<i>val</i>	fuse write value

The command is passed as is to SECO. SECO uses part of the *word* parameter to indicate if the fuse should be locked after programming. See the "Write common fuse" section of the Security Reference Manual (SRM) for more info.

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if word fuse index param out of range or invalid
- SC_ERR_NOACCESS if write operation failed
- SC_ERR_LOCKED if write operation is locked

10.2.2.28 `sc_misc_set_temp()`

```

sc_err_t sc_misc_set_temp (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_misc_temp_t temp,
    int16_t celsius,
    int8_t tenths )

```

This function sets a temp sensor alarm.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource with sensor
in	<i>temp</i>	alarm to set
in	<i>celsius</i>	whole part of temp to set
in	<i>tenths</i>	fractional part of temp to set

Returns

Returns and error code (SC_ERR_NONE = success).

This function will enable the alarm interrupt if the temp requested is not the min/max temp. This enable automatically clears when the alarm occurs and this function has to be called again to re-enable.

Return errors codes:

- SC_ERR_PARM if parameters invalid

10.2.2.29 `sc_misc_get_temp()`

```

sc_err_t sc_misc_get_temp (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_misc_temp_t temp,
    int16_t * celsius,
    int8_t * tenths )

```

This function gets a temp sensor value.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource with sensor
in	<i>temp</i>	value to get (sensor or alarm)
out	<i>celsius</i>	whole part of temp to get
out	<i>tenths</i>	fractional part of temp to get

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_PARM if parameters invalid
- SC_ERR_BUSY if temp not ready yet (time delay after power on)

10.2.2.30 sc_misc_get_boot_dev()

```
void sc_misc_get_boot_dev (
    sc_ipc_t ipc,
    sc_rsrc_t * dev )
```

This function returns the boot device.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>dev</i>	pointer to return boot device

10.2.2.31 sc_misc_get_boot_type()

```
sc_err_t sc_misc_get_boot_type (
    sc_ipc_t ipc,
    sc_misc_bt_t * type )
```

This function returns the boot type.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>type</i>	pointer to return boot type

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors code:

- SC_ERR_UNAVAILABLE if type not passed by ROM

10.2.2.32 sc_misc_get_button_status()

```
void sc_misc_get_button_status (
    sc_ipc_t ipc,
    sc_bool_t * status )
```

This function returns the current status of the ON/OFF button.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>status</i>	pointer to return button status

10.2.2.33 sc_misc_rompatch_checksum()

```
sc_err_t sc_misc_rompatch_checksum (
    sc_ipc_t ipc,
    uint32_t * checksum )
```

This function returns the ROM patch checksum.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>checksum</i>	pointer to return checksum

Returns

Returns and error code (SC_ERR_NONE = success).

10.3 (SVC) Pad Service

Module for the Pad Control (PAD) service.

Typedefs

- typedef [uint8_t sc_pad_config_t](#)
This type is used to declare a pad config.
- typedef [uint8_t sc_pad_iso_t](#)
This type is used to declare a pad low-power isolation config.
- typedef [uint8_t sc_pad_28fdsoi_dse_t](#)
This type is used to declare a drive strength.
- typedef [uint8_t sc_pad_28fdsoi_ps_t](#)
This type is used to declare a pull select.
- typedef [uint8_t sc_pad_28fdsoi_pus_t](#)
This type is used to declare a pull-up select.
- typedef [uint8_t sc_pad_wakeup_t](#)
This type is used to declare a wakeup mode of a pad.

Defines for type widths

- #define [SC_PAD_MUX_W](#) 3U
Width of mux parameter.

Defines for [sc_pad_config_t](#)

- #define [SC_PAD_CONFIG_NORMAL](#) 0U
Normal.
- #define [SC_PAD_CONFIG_OD](#) 1U
Open Drain.
- #define [SC_PAD_CONFIG_OD_IN](#) 2U
Open Drain and input.
- #define [SC_PAD_CONFIG_OUT_IN](#) 3U
Output and input.

Defines for [sc_pad_iso_t](#)

- #define [SC_PAD_ISO_OFF](#) 0U
ISO latch is transparent.
- #define [SC_PAD_ISO_EARLY](#) 1U
Follow EARLY_ISO.
- #define [SC_PAD_ISO_LATE](#) 2U
Follow LATE_ISO.
- #define [SC_PAD_ISO_ON](#) 3U
ISO latched data is held.

Defines for `sc_pad_28fdsoi_dse_t`

- `#define SC_PAD_28FDSOI_DSE_18V_1MA 0U`
Drive strength of 1mA for 1.8v.
- `#define SC_PAD_28FDSOI_DSE_18V_2MA 1U`
Drive strength of 2mA for 1.8v.
- `#define SC_PAD_28FDSOI_DSE_18V_4MA 2U`
Drive strength of 4mA for 1.8v.
- `#define SC_PAD_28FDSOI_DSE_18V_6MA 3U`
Drive strength of 6mA for 1.8v.
- `#define SC_PAD_28FDSOI_DSE_18V_8MA 4U`
Drive strength of 8mA for 1.8v.
- `#define SC_PAD_28FDSOI_DSE_18V_10MA 5U`
Drive strength of 10mA for 1.8v.
- `#define SC_PAD_28FDSOI_DSE_18V_12MA 6U`
Drive strength of 12mA for 1.8v.
- `#define SC_PAD_28FDSOI_DSE_18V_HS 7U`
High-speed drive strength for 1.8v.
- `#define SC_PAD_28FDSOI_DSE_33V_2MA 0U`
Drive strength of 2mA for 3.3v.
- `#define SC_PAD_28FDSOI_DSE_33V_4MA 1U`
Drive strength of 4mA for 3.3v.
- `#define SC_PAD_28FDSOI_DSE_33V_8MA 2U`
Drive strength of 8mA for 3.3v.
- `#define SC_PAD_28FDSOI_DSE_33V_12MA 3U`
Drive strength of 12mA for 3.3v.
- `#define SC_PAD_28FDSOI_DSE_DV_HIGH 0U`
High drive strength for dual volt.
- `#define SC_PAD_28FDSOI_DSE_DV_LOW 1U`
Low drive strength for dual volt.

Defines for `sc_pad_28fdsoi_ps_t`

- `#define SC_PAD_28FDSOI_PS_KEEPER 0U`
Bus-keeper (only valid for 1.8v)
- `#define SC_PAD_28FDSOI_PS_PU 1U`
Pull-up.
- `#define SC_PAD_28FDSOI_PS_PD 2U`
Pull-down.
- `#define SC_PAD_28FDSOI_PS_NONE 3U`
No pull (disabled)

Defines for `sc_pad_28fdsoi_pus_t`

- `#define SC_PAD_28FDSOI_PUS_30K_PD 0U`
30K pull-down
- `#define SC_PAD_28FDSOI_PUS_100K_PU 1U`
100K pull-up
- `#define SC_PAD_28FDSOI_PUS_3K_PU 2U`
3K pull-up
- `#define SC_PAD_28FDSOI_PUS_30K_PU 3U`
30K pull-up

Defines for `sc_pad_wakeup_t`

- `#define SC_PAD_WAKEUP_OFF 0U`
Off.
- `#define SC_PAD_WAKEUP_CLEAR 1U`
Clears pending flag.
- `#define SC_PAD_WAKEUP_LOW_LVL 4U`
Low level.
- `#define SC_PAD_WAKEUP_FALL_EDGE 5U`
Falling edge.
- `#define SC_PAD_WAKEUP_RISE_EDGE 6U`
Rising edge.
- `#define SC_PAD_WAKEUP_HIGH_LVL 7U`
High-level.

Generic Functions

- `sc_err_t sc_pad_set_mux` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint8_t mux`, `sc_pad_config_t config`, `sc_pad_iso_t iso`)
This function configures the mux settings for a pad.
- `sc_err_t sc_pad_get_mux` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint8_t *mux`, `sc_pad_config_t *config`, `sc_pad_iso_t *iso`)
This function gets the mux settings for a pad.
- `sc_err_t sc_pad_set_gp` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint32_t ctrl`)
This function configures the general purpose pad control.
- `sc_err_t sc_pad_get_gp` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint32_t *ctrl`)
This function gets the general purpose pad control.
- `sc_err_t sc_pad_set_wakeup` (`sc_ipc_t ipc`, `sc_pad_t pad`, `sc_pad_wakeup_t wakeup`)
This function configures the wakeup mode of the pad.
- `sc_err_t sc_pad_get_wakeup` (`sc_ipc_t ipc`, `sc_pad_t pad`, `sc_pad_wakeup_t *wakeup`)
This function gets the wakeup mode of a pad.
- `sc_err_t sc_pad_set_all` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint8_t mux`, `sc_pad_config_t config`, `sc_pad_iso_t iso`, `uint32_t ctrl`, `sc_pad_wakeup_t wakeup`)
This function configures a pad.
- `sc_err_t sc_pad_get_all` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint8_t *mux`, `sc_pad_config_t *config`, `sc_pad_iso_t *iso`, `uint32_t *ctrl`, `sc_pad_wakeup_t *wakeup`)
This function gets a pad's config.

SoC Specific Functions

- `sc_err_t sc_pad_set` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint32_t val`)
This function configures the settings for a pad.
- `sc_err_t sc_pad_get` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint32_t *val`)
This function gets the settings for a pad.

Technology Specific Functions

- `sc_err_t sc_pad_set_gp_28fdsoi` (`sc_ipc_t ipc`, `sc_pad_t pad`, `sc_pad_28fdsoi_dse_t dse`, `sc_pad_28fdsoi_ps_t ps`)
This function configures the pad control specific to 28FDSOI.
- `sc_err_t sc_pad_get_gp_28fdsoi` (`sc_ipc_t ipc`, `sc_pad_t pad`, `sc_pad_28fdsoi_dse_t *dse`, `sc_pad_28fdsoi_ps_t *ps`)
This function gets the pad control specific to 28FDSOI.
- `sc_err_t sc_pad_set_gp_28fdsoi_hsic` (`sc_ipc_t ipc`, `sc_pad_t pad`, `sc_pad_28fdsoi_dse_t dse`, `sc_bool_t hys`, `sc_pad_28fdsoi_pus_t pus`, `sc_bool_t pke`, `sc_bool_t pue`)
This function configures the pad control specific to 28FDSOI.
- `sc_err_t sc_pad_get_gp_28fdsoi_hsic` (`sc_ipc_t ipc`, `sc_pad_t pad`, `sc_pad_28fdsoi_dse_t *dse`, `sc_bool_t *hys`, `sc_pad_28fdsoi_pus_t *pus`, `sc_bool_t *pke`, `sc_bool_t *pue`)
This function gets the pad control specific to 28FDSOI.
- `sc_err_t sc_pad_set_gp_28fdsoi_comp` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint8_t compen`, `sc_bool_t fastfrz`, `uint8_t rasrcp`, `uint8_t rasrcn`, `sc_bool_t nasrc_sel`, `sc_bool_t psw_ovr`)
This function configures the compensation control specific to 28FDSOI.
- `sc_err_t sc_pad_get_gp_28fdsoi_comp` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint8_t *compen`, `sc_bool_t *fastfrz`, `uint8_t *rasrcp`, `uint8_t *rasrcn`, `sc_bool_t *nasrc_sel`, `sc_bool_t *compok`, `uint8_t *nasrc`, `sc_bool_t *psw_ovr`)
This function gets the compensation control specific to 28FDSOI.

10.3.1 Detailed Description

Module for the Pad Control (PAD) service.

Pad configuration is managed by SC firmware. The pad configuration features supported by the SC firmware include:

- Configuring the mux, input/output connection, and low-power isolation mode.
- Configuring the technology-specific pad setting such as drive strength, pullup/pulldown, etc.
- Configuring compensation for pad groups with dual voltage capability.

Pad functions fall into one of three categories. Generic functions are common to all SoCs and all process technologies. SoC functions are raw low-level functions. Technology-specific functions are specific to the process technology.

The list of pads is SoC specific. Refer to the SoC [Pad List](#) for valid pad values. Note that all pads exist on a die but may or may not be brought out by the specific package. Mapping of pads to package pins/balls is documented in the associated Data Sheet. Some pads may not be brought out because the part (die+package) is defeatured and some pads may connect to the substrate in the package.

Some pads (SC_P_COMP_*) that can be specified are not individual pads but are in fact pad groups. These groups have additional configuration that can be done using the `sc_pad_set_gp_28fdsoi_comp()` function. More info on these can be found in the associated Reference Manual.

Pads are managed as a resource by the Resource Manager (RM). They have assigned owners and only the owners can configure the pads. Some of the pads are reserved for use by the SCFW itself and this can be overridden with the implementation of `board_config_sc()`. Additionally, pads may be assigned to various other partitions via the implementation of `board_system_config()`.

Note muxing two input pads to the same IP functional signal will result in undefined behavior.

The following SCFW pad code is an example of how to configure pads. In this example, two pads are configured for use by the i.MX8QXP I2C_0 (ADMA.I2C0). Another dual-voltage pad is configured as SPI_0 SCK (ADMA.SPI0.SCK).

The `ipc` parameter most functions take is a handle to the IPC channel opened to communicate to the SC. It is implementation defined. Most API ports include an `sc_ipc_open()` and `sc_ipc_close()` function to manage this. The `sc_ipc_open()` takes an argument to identify the communication channel (usually the MU address) and returns the IPC handle that all API calls should then use.

```
1 /* Configure I2C_0 SCL pad */
2 sc_pad_set_mux(ipc, SC_P_MIPI_CSI0_GPIO0_00, 1, SC_PAD_CONFIG_OD_IN, SC_PAD_ISO_OFF);
3 sc_pad_set_gp_28fdsoi(ipc, SC_P_MIPI_CSI0_GPIO0_00, SC_PAD_28FDSOI_DSE_18V_1MA, SC_PAD_28FDSOI_PS_PU);
4
5 /* Configure I2C_0 SDA pad */
6 sc_pad_set_mux(ipc, SC_P_MIPI_CSI0_GPIO0_01, 1, SC_PAD_CONFIG_OD_IN, SC_PAD_ISO_OFF);
7 sc_pad_set_gp_28fdsoi(ipc, SC_P_MIPI_CSI0_GPIO0_01, SC_PAD_28FDSOI_DSE_18V_1MA, SC_PAD_28FDSOI_PS_PU);
8
9 /* Configure SPI0 SCK pad (dual-voltage) */
10 sc_pad_set_mux(ipc, SC_P_SPI0_SCK, 0, SC_PAD_CONFIG_NORMAL, SC_PAD_ISO_OFF);
11 sc_pad_set_gp_28fdsoi(ipc, SC_P_SPI0_SCK, SC_PAD_28FDSOI_DSE_DV_LOW, SC_PAD_28FDSOI_PS_NONE);
```

The first pair of pads in question are MIPI_CSI0_GPIO0_00 (used for SCL) and MIPI_CSI0_GPIO0_01 (used for SDA). I2C_0 is mux select 1 for both pads.

The first two lines configure the SCL pad. The first configures the SCL pad for mux select 1, and as open-drain with with input. The second configures the drive strength and enables the pull-up.

The last two lines do the same for the SDA pad.

For 28FDSIO single voltage pads, SC_PAD_28FDSOI_DSE_DV_HIGH and SC_PAD_28FDSOI_DSE_DV_LOW are not valid drive strenths.

```
/* Configure I2C_0 SCL pad */
sc_pad_set_mux(ipc, SC_P_MIPI_CSI0_GPIO0_00, 1, SC_PAD_CONFIG_OD_IN, SC_PAD_ISO_OFF);
sc_pad_set_gp_28fdsoi(ipc, SC_P_MIPI_CSI0_GPIO0_00, SC_PAD_28FDSOI_DSE_18V_1MA, SC_PAD_28FDSOI_PS_PU);
/* Configure I2C_0 SDA pad */
sc_pad_set_mux(ipc, SC_P_MIPI_CSI0_GPIO0_01, 1, SC_PAD_CONFIG_OD_IN, SC_PAD_ISO_OFF);
sc_pad_set_gp_28fdsoi(ipc, SC_P_MIPI_CSI0_GPIO0_01, SC_PAD_28FDSOI_DSE_18V_1MA, SC_PAD_28FDSOI_PS_PU);
```

The next pad configured is SPI0_SCK. It is configured as mux select 0. the first line configures the mux select as 0 and normal push-pull. The second line configures the drive strength and no pull-up. Note the drive strength setting is different for this dual voltage pad.

```
/* Configure SPI0 SCK pad (dual-voltage) */
sc_pad_set_mux(ipc, SC_P_SPI0_SCK, 0, SC_PAD_CONFIG_NORMAL, SC_PAD_ISO_OFF);
sc_pad_set_gp_28fdsoi(ipc, SC_P_SPI0_SCK, SC_PAD_28FDSOI_DSE_DV_LOW, SC_PAD_28FDSOI_PS_NONE);
```

For 28FDSIO dual voltage pads, only SC_PAD_28FDSOI_DSE_DV_HIGH and SC_PAD_28FDSOI_DSE_DV_LOW are valid drive strenths.

The voltage of the pad is determined by the supply for the pad group (the VDD_SPI_SAI_1P8_3P3 pad in this case).

10.3.2 Typedef Documentation

10.3.2.1 sc_pad_config_t

```
typedef uint8_t sc_pad_config_t
```

This type is used to declare a pad config.

It determines how the output data is driven, pull-up is controlled, and input signal is connected. Normal and OD are typical and only connect the input when the output is not driven. The IN options are less common and force an input connection even when driving the output.

10.3.2.2 sc_pad_iso_t

```
typedef uint8_t sc_pad_iso_t
```

This type is used to declare a pad low-power isolation config.

ISO_LATE is the most common setting. ISO_EARLY is only used when an output pad is directly determined by another input pad. The other two are only used when SW wants to directly control isolation.

10.3.2.3 sc_pad_28fdsoi_dse_t

```
typedef uint8_t sc_pad_28fdsoi_dse_t
```

This type is used to declare a drive strength.

Note it is specific to 28FDSOI. Also note that valid values depend on the pad type.

10.3.2.4 sc_pad_28fdsoi_ps_t

```
typedef uint8_t sc_pad_28fdsoi_ps_t
```

This type is used to declare a pull select.

Note it is specific to 28FDSOI.

10.3.2.5 sc_pad_28fdsoi_pus_t

```
typedef uint8_t sc_pad_28fdsoi_pus_t
```

This type is used to declare a pull-up select.

Note it is specific to 28FDSOI HSIC pads.

10.3.3 Function Documentation

10.3.3.1 `sc_pad_set_mux()`

```
sc_err_t sc_pad_set_mux (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint8_t mux,
    sc_pad_config_t config,
    sc_pad_iso_t iso )
```

This function configures the mux settings for a pad.

This includes the signal mux, pad config, and low-power isolation mode.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to configure
in	<i>mux</i>	mux setting
in	<i>config</i>	pad config
in	<i>iso</i>	low-power isolation mode

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Note muxing two input pads to the same IP functional signal will result in undefined behavior.

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.2 `sc_pad_get_mux()`

```
sc_err_t sc_pad_get_mux (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint8_t * mux,
    sc_pad_config_t * config,
    sc_pad_iso_t * iso )
```

This function gets the mux settings for a pad.

This includes the signal mux, pad config, and low-power isolation mode.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to query
out	<i>mux</i>	pointer to return mux setting
out	<i>config</i>	pointer to return pad config
out	<i>iso</i>	pointer to return low-power isolation mode

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.3 sc_pad_set_gp()

```
sc_err_t sc_pad_set_gp (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint32_t ctrl )
```

This function configures the general purpose pad control.

This is technology dependent and includes things like drive strength, slew rate, pull up/down, etc. Refer to the SoC Reference Manual for bit field details.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to configure
in	<i>ctrl</i>	control value to set

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.4 sc_pad_get_gp()

```
sc_err_t sc_pad_get_gp (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint32_t * ctrl )
```

This function gets the general purpose pad control.

This is technology dependent and includes things like drive strength, slew rate, pull up/down, etc. Refer to the SoC Reference Manual for bit field details.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to query
out	<i>ctrl</i>	pointer to return control value

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.5 sc_pad_set_wakeup()

```
sc_err_t sc_pad_set_wakeup (
    sc_ipc_t ipc,
    sc_pad_t pad,
    sc_pad_wakeup_t wakeup )
```

This function configures the wakeup mode of the pad.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to configure
in	<i>wakeup</i>	wakeup to set

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.6 sc_pad_get_wakeup()

```
sc_err_t sc_pad_get_wakeup (
    sc_ipc_t ipc,
    sc_pad_t pad,
    sc_pad_wakeup_t * wakeup )
```

This function gets the wakeup mode of a pad.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to query
out	<i>wakeup</i>	pointer to return wakeup

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.7 sc_pad_set_all()

```
sc_err_t sc_pad_set_all (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint8_t mux,
    sc_pad_config_t config,
    sc_pad_iso_t iso,
    uint32_t ctrl,
    sc_pad_wakeup_t wakeup )
```

This function configures a pad.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to configure
in	<i>mux</i>	mux setting
in	<i>config</i>	pad config
in	<i>iso</i>	low-power isolation mode
in	<i>ctrl</i>	control value
in	<i>wakeup</i>	wakeup to set

See also

[sc_pad_set_mux\(\)](#).
[sc_pad_set_gp\(\)](#).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Returns

Returns an error code (SC_ERR_NONE = success).

Note muxing two input pads to the same IP functional signal will result in undefined behavior.

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.8 sc_pad_get_all()

```
sc_err_t sc_pad_get_all (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint8_t * mux,
    sc_pad_config_t * config,
    sc_pad_iso_t * iso,
    uint32_t * ctrl,
    sc_pad_wakeup_t * wakeup )
```

This function gets a pad's config.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to query
out	<i>mux</i>	pointer to return mux setting
out	<i>config</i>	pointer to return pad config
out	<i>iso</i>	pointer to return low-power isolation mode
out	<i>ctrl</i>	pointer to return control value
out	<i>wakeup</i>	pointer to return wakeup to set

See also

[sc_pad_set_mux\(\)](#).
[sc_pad_set_gp\(\)](#).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Returns

Returns an error code (SC_ERR_NONE = success).

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.9 sc_pad_set()

```
sc_err_t sc_pad_set (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint32_t val )
```

This function configures the settings for a pad.

This setting is SoC specific.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to configure
in	<i>val</i>	value to set

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.10 sc_pad_get()

```
sc_err_t sc_pad_get (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint32_t * val )
```

This function gets the settings for a pad.

This setting is SoC specific.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to query
out	<i>val</i>	pointer to return setting

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.11 sc_pad_set_gp_28fdsoi()

```
sc_err_t sc_pad_set_gp_28fdsoi (
    sc_ipc_t ipc,
    sc_pad_t pad,
    sc_pad_28fdsoi_dse_t dse,
    sc_pad_28fdsoi_ps_t ps )
```

This function configures the pad control specific to 28FDSOI.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to configure
in	<i>dse</i>	drive strength
in	<i>ps</i>	pull select

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner,
- SC_ERR_UNAVAILABLE if process not applicable

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.12 sc_pad_get_gp_28fdsoi()

```
sc_err_t sc_pad_get_gp_28fdsoi (
    sc_ipc_t ipc,
    sc_pad_t pad,
    sc_pad_28fdsoi_dse_t * dse,
    sc_pad_28fdsoi_ps_t * ps )
```

This function gets the pad control specific to 28FDSOI.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to query
out	<i>dse</i>	pointer to return drive strength
out	<i>ps</i>	pointer to return pull select

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner,
- SC_ERR_UNAVAILABLE if process not applicable

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.13 sc_pad_set_gp_28fdsoi_hsic()

```
sc_err_t sc_pad_set_gp_28fdsoi_hsic (
    sc_ipc_t ipc,
    sc_pad_t pad,
    sc_pad_28fdsoi_dse_t dse,
    sc_bool_t hys,
    sc_pad_28fdsoi_pus_t pus,
    sc_bool_t pke,
    sc_bool_t pue )
```

This function configures the pad control specific to 28FDSOI.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to configure
in	<i>dse</i>	drive strength
in	<i>hys</i>	hysteresis
in	<i>pus</i>	pull-up select
in	<i>pke</i>	pull keeper enable
in	<i>pue</i>	pull-up enable

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner,
- SC_ERR_UNAVAILABLE if process not applicable

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.14 sc_pad_get_gp_28fdsoi_hsic()

```
sc_err_t sc_pad_get_gp_28fdsoi_hsic (
    sc_ipc_t ipc,
    sc_pad_t pad,
    sc_pad_28fdsoi_dse_t * dse,
    sc_bool_t * hys,
    sc_pad_28fdsoi_pus_t * pus,
    sc_bool_t * pke,
    sc_bool_t * pue )
```

This function gets the pad control specific to 28FDSOI.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to query
out	<i>dse</i>	pointer to return drive strength
out	<i>hys</i>	pointer to return hysteresis
out	<i>pus</i>	pointer to return pull-up select
out	<i>pke</i>	pointer to return pull keeper enable
out	<i>pue</i>	pointer to return pull-up enable

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner,
- SC_ERR_UNAVAILABLE if process not applicable

Refer to the SoC [Pad List](#) for valid pad values.

10.3.3.15 sc_pad_set_gp_28fdsoi_comp()

```
sc_err_t sc_pad_set_gp_28fdsoi_comp (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint8_t compen,
    sc_bool_t fastfrz,
    uint8_t rasrcp,
    uint8_t rasrcn,
    sc_bool_t nasrc_sel,
    sc_bool_t psw_ovr )
```

This function configures the compensation control specific to 28FDSOI.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to configure
in	<i>compen</i>	compensation/freeze mode
in	<i>fastfrz</i>	fast freeze
in	<i>rasrcp</i>	compensation code for PMOS
in	<i>rasrcn</i>	compensation code for NMOS
in	<i>nasrc_sel</i>	NASRC read select
in	<i>psw_ovr</i>	2.5v override

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner,
- SC_ERR_UNAVAILABLE if process not applicable

Refer to the SoC [Pad List](#) for valid pad values.

Note *psw_ovr* is only applicable to pads supporting 2.5 volt operation (e.g. some Ethernet pads).

10.3.3.16 sc_pad_get_gp_28fdsoi_comp()

```
sc_err_t sc_pad_get_gp_28fdsoi_comp (
    sc_ipc_t ipc,
    sc_pad_t pad,
    uint8_t * compen,
    sc_bool_t * fastfrz,
    uint8_t * rasrcp,
    uint8_t * rasrcn,
    sc_bool_t * nasrc_sel,
    sc_bool_t * compok,
    uint8_t * nasrc,
    sc_bool_t * psw_ovr )
```

This function gets the compensation control specific to 28FDSOI.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to query
out	<i>compen</i>	pointer to return compensation/freeze mode
out	<i>fastfrz</i>	pointer to return fast freeze
out	<i>rasrcp</i>	pointer to return compensation code for PMOS
out	<i>rasrcn</i>	pointer to return compensation code for NMOS
out	<i>nasrc_sel</i>	pointer to return NASRC read select
out	<i>compok</i>	pointer to return compensation status
out	<i>nasrc</i>	pointer to return NASRCP/NASRCN
out	<i>psw_ovr</i>	pointer to return the 2.5v override

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner,
- SC_ERR_UNAVAILABLE if process not applicable

Refer to the SoC [Pad List](#) for valid pad values.

10.4 (SVC) Power Management Service

Module for the Power Management (PM) service.

Typedefs

- typedef [uint8_t sc_pm_power_mode_t](#)
This type is used to declare a power mode.
- typedef [uint8_t sc_pm_clk_t](#)
This type is used to declare a clock.
- typedef [uint8_t sc_pm_clk_mode_t](#)
This type is used to declare a clock mode.
- typedef [uint8_t sc_pm_clk_parent_t](#)
This type is used to declare the clock parent.
- typedef [uint32_t sc_pm_clock_rate_t](#)
This type is used to declare clock rates.
- typedef [uint8_t sc_pm_reset_type_t](#)
This type is used to declare a desired reset type.
- typedef [uint8_t sc_pm_reset_reason_t](#)
This type is used to declare a reason for a reset.
- typedef [uint8_t sc_pm_sys_if_t](#)
This type is used to specify a system-level interface to be power managed.
- typedef [uint8_t sc_pm_wake_src_t](#)
This type is used to specify a wake source for CPU resources.

Defines for type widths

- #define [SC_PM_POWER_MODE_W](#) 2U
Width of [sc_pm_power_mode_t](#).
- #define [SC_PM_CLOCK_MODE_W](#) 3U
Width of [sc_pm_clock_mode_t](#).
- #define [SC_PM_RESET_TYPE_W](#) 2U
Width of [sc_pm_reset_type_t](#).
- #define [SC_PM_RESET_REASON_W](#) 4U
Width of [sc_pm_reset_reason_t](#).

Defines for ALL parameters

- #define [SC_PM_CLK_ALL](#) (([sc_pm_clk_t](#)) UINT8_MAX)
All clocks.

Defines for `sc_pm_power_mode_t`

- `#define SC_PM_PW_MODE_OFF 0U`
Power off.
- `#define SC_PM_PW_MODE_STBY 1U`
Power in standby.
- `#define SC_PM_PW_MODE_LP 2U`
Power in low-power.
- `#define SC_PM_PW_MODE_ON 3U`
Power on.

Defines for `sc_pm_clk_t`

- `#define SC_PM_CLK_SLV_BUS 0U`
Slave bus clock.
- `#define SC_PM_CLK_MST_BUS 1U`
Master bus clock.
- `#define SC_PM_CLK_PER 2U`
Peripheral clock.
- `#define SC_PM_CLK_PHY 3U`
Phy clock.
- `#define SC_PM_CLK_MISC 4U`
Misc clock.
- `#define SC_PM_CLK_MISC0 0U`
Misc 0 clock.
- `#define SC_PM_CLK_MISC1 1U`
Misc 1 clock.
- `#define SC_PM_CLK_MISC2 2U`
Misc 2 clock.
- `#define SC_PM_CLK_MISC3 3U`
Misc 3 clock.
- `#define SC_PM_CLK_MISC4 4U`
Misc 4 clock.
- `#define SC_PM_CLK_CPU 2U`
CPU clock.
- `#define SC_PM_CLK_PLL 4U`
PLL.
- `#define SC_PM_CLK_BYPASS 4U`
Bypass clock.

Defines for `sc_pm_clk_mode_t`

- `#define SC_PM_CLK_MODE_ROM_INIT 0U`
Clock is initialized by ROM.
- `#define SC_PM_CLK_MODE_OFF 1U`
Clock is disabled.
- `#define SC_PM_CLK_MODE_ON 2U`
Clock is enabled.
- `#define SC_PM_CLK_MODE_AUTOGATE_SW 3U`
Clock is in SW autogate mode.
- `#define SC_PM_CLK_MODE_AUTOGATE_HW 4U`
Clock is in HW autogate mode.
- `#define SC_PM_CLK_MODE_AUTOGATE_SW_HW 5U`
Clock is in SW-HW autogate mode.

Defines for `sc_pm_clk_parent_t`

- `#define SC_PM_PARENT_XTAL 0U`
Parent is XTAL.
- `#define SC_PM_PARENT_PLL0 1U`
Parent is PLL0.
- `#define SC_PM_PARENT_PLL1 2U`
Parent is PLL1 or PLL0/2.
- `#define SC_PM_PARENT_PLL2 3U`
Parent in PLL2 or PLL0/4.
- `#define SC_PM_PARENT_BYPASS 4U`
Parent is a bypass clock.

Defines for `sc_pm_reset_type_t`

- `#define SC_PM_RESET_TYPE_COLD 0U`
Cold reset.
- `#define SC_PM_RESET_TYPE_WARM 1U`
Warm reset.
- `#define SC_PM_RESET_TYPE_BOARD 2U`
Board reset.

Defines for `sc_pm_reset_reason_t`

- `#define SC_PM_RESET_REASON_POR 0U`
Power on reset.
- `#define SC_PM_RESET_REASON_JTAG 1U`
JTAG reset.
- `#define SC_PM_RESET_REASON_SW 2U`
Software reset.
- `#define SC_PM_RESET_REASON_WDOG 3U`
Partition watchdog reset.
- `#define SC_PM_RESET_REASON_LOCKUP 4U`
SCU lockup reset.
- `#define SC_PM_RESET_REASON_SNVS 5U`
SNVS reset.
- `#define SC_PM_RESET_REASON_TEMP 6U`
Temp panic reset.
- `#define SC_PM_RESET_REASON_MSI 7U`
MSI reset.
- `#define SC_PM_RESET_REASON_UECC 8U`
ECC reset.
- `#define SC_PM_RESET_REASON_SCFW_WDOG 9U`
SCFW watchdog reset.
- `#define SC_PM_RESET_REASON_ROM_WDOG 10U`
SCU ROM watchdog reset.
- `#define SC_PM_RESET_REASON_SECO 11U`
SECO reset.
- `#define SC_PM_RESET_REASON_SCFW_FAULT 12U`
SCFW fault reset.

Defines for `sc_pm_sys_if_t`

- `#define SC_PM_SYS_IF_INTERCONNECT 0U`
System interconnect.
- `#define SC_PM_SYS_IF_MU 1U`
AP -> SCU message units.
- `#define SC_PM_SYS_IF_OCMEM 2U`
On-chip memory (ROM/OCRAM)
- `#define SC_PM_SYS_IF_DDR 3U`
DDR memory.

Defines for `sc_pm_wake_src_t`

- `#define SC_PM_WAKE_SRC_NONE 0U`
No wake source, used for self-kill.
- `#define SC_PM_WAKE_SRC_SCU 1U`
Wakeup from SCU to resume CPU (IRQSTEER & GIC powered down)
- `#define SC_PM_WAKE_SRC_IRQSTEER 2U`
Wakeup from IRQSTEER to resume CPU (GIC powered down)
- `#define SC_PM_WAKE_SRC_IRQSTEER_GIC 3U`
Wakeup from IRQSTEER+GIC to wake CPU (GIC clock gated)
- `#define SC_PM_WAKE_SRC_GIC 4U`
Wakeup from GIC to wake CPU.

Power Functions

- `sc_err_t sc_pm_set_sys_power_mode (sc_ipc_t ipc, sc_pm_power_mode_t mode)`
This function sets the system power mode.
- `sc_err_t sc_pm_set_partition_power_mode (sc_ipc_t ipc, sc_rm_pt_t pt, sc_pm_power_mode_t mode)`
This function sets the power mode of a partition.
- `sc_err_t sc_pm_get_sys_power_mode (sc_ipc_t ipc, sc_rm_pt_t pt, sc_pm_power_mode_t *mode)`
This function gets the power mode of a partition.
- `sc_err_t sc_pm_set_resource_power_mode (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_power_mode_t mode)`
This function sets the power mode of a resource.
- `sc_err_t sc_pm_set_resource_power_mode_all (sc_ipc_t ipc, sc_rm_pt_t pt, sc_pm_power_mode_t mode, sc_rsrc_t exclude)`
This function sets the power mode for all the resources owned by a child partition.
- `sc_err_t sc_pm_get_resource_power_mode (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_power_mode_t *mode)`
This function gets the power mode of a resource.
- `sc_err_t sc_pm_req_low_power_mode (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_power_mode_t mode)`
This function requests the low power mode some of the resources can enter based on their state.
- `sc_err_t sc_pm_req_cpu_low_power_mode (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_power_mode_t mode, sc_pm_wake_src_t wake_src)`
This function requests low-power mode entry for CPU/cluster resources.
- `sc_err_t sc_pm_set_cpu_resume_addr (sc_ipc_t ipc, sc_rsrc_t resource, sc_faddr_t address)`
This function is used to set the resume address of a CPU.
- `sc_err_t sc_pm_set_cpu_resume (sc_ipc_t ipc, sc_rsrc_t resource, sc_bool_t isPrimary, sc_faddr_t address)`
This function is used to set parameters for CPU resume from low-power mode.
- `sc_err_t sc_pm_req_sys_if_power_mode (sc_ipc_t ipc, sc_rsrc_t resource, sc_pm_sys_if_t sys_if, sc_pm_power_mode_t hpm, sc_pm_power_mode_t lpm)`
This function requests the power mode configuration for system-level interfaces including messaging units, interconnect, and memories.

Clock/PLL Functions

- `sc_err_t sc_pm_set_clock_rate` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_pm_clk_t` clk, `sc_pm_clock_rate_t` *rate)
This function sets the rate of a resource's clock/PLL.
- `sc_err_t sc_pm_get_clock_rate` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_pm_clk_t` clk, `sc_pm_clock_rate_t` *rate)
This function gets the rate of a resource's clock/PLL.
- `sc_err_t sc_pm_clock_enable` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_pm_clk_t` clk, `sc_bool_t` enable, `sc_bool_t` autog)
This function enables/disables a resource's clock.
- `sc_err_t sc_pm_set_clock_parent` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_pm_clk_t` clk, `sc_pm_clk_parent_t` parent)
This function sets the parent of a resource's clock.
- `sc_err_t sc_pm_get_clock_parent` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_pm_clk_t` clk, `sc_pm_clk_parent_t` *parent)
This function gets the parent of a resource's clock.

Reset Functions

- `sc_err_t sc_pm_reset` (`sc_ipc_t` ipc, `sc_pm_reset_type_t` type)
This function is used to reset the system.
- `sc_err_t sc_pm_reset_reason` (`sc_ipc_t` ipc, `sc_pm_reset_reason_t` *reason)
This function gets a caller's reset reason.
- `sc_err_t sc_pm_boot` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_rsrc_t` resource_cpu, `sc_faddr_t` boot_addr, `sc_rsrc_t` resource_mu, `sc_rsrc_t` resource_dev)
This function is used to boot a partition.
- `void sc_pm_reboot` (`sc_ipc_t` ipc, `sc_pm_reset_type_t` type)
This function is used to reboot the caller's partition.
- `sc_err_t sc_pm_reboot_partition` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_pm_reset_type_t` type)
This function is used to reboot a partition.
- `sc_err_t sc_pm_cpu_start` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_bool_t` enable, `sc_faddr_t` address)
This function is used to start/stop a CPU.

10.4.1 Detailed Description

Module for the Power Management (PM) service.

The following SCFW PM code is an example of how to configure the power and clocking of a UART. All resources MUST be powered on before accessing.

The ipc parameter most functions take is a handle to the IPC channel opened to communicate to the SC. It is implementation defined. Most API ports include an `sc_ipc_open()` and `sc_ipc_close()` function to manage this. The `sc_ipc_open()` takes an argument to identify the communication channel (usually the MU address) and returns the IPC handle that all API calls should then use.

Refer to the SoC-specific [Resource List](#) for a list of resources. Refer to the SoC-specific [Clock List](#) for a list of clocks.

```
1 sc_pm_clock_rate_t rate = SC_160MHZ;
```

```

2
3 /* Powerup UART 0 */
4 sc_pm_set_resource_power_mode(ipc, SC_R_UART_0,          SC_PM_PW_MODE_ON);
5
6 /* Configure UART 0 baud clock */
7 sc_pm_set_clock_rate(ipc, SC_R_UART_0, SC_PM_CLK_PER, &rate);
8
9 /* Enable UART 0 clock */
10 sc_pm_clock_enable(ipc, SC_R_UART_0, SC_PM_CLK_PER,      SC_TRUE, SC_FALSE);

```

First, a variable is declared to hold the rate to request and return for the UART peripheral clock. Note this is the baud clock going into the UART which is then further divided within the UART itself.

```
sc_pm_clock_rate_t rate = SC_160MHZ;
```

Then change the power state of the UART to the ON state.

```

/* Powerup UART 0 */
sc_pm_set_resource_power_mode(ipc, SC_R_UART_0,          SC_PM_PW_MODE_ON);

```

Then configure the UART peripheral clock. Note that due to hardware limitation, the exact rate may not be what is requested. The rate is guaranteed to not be greater than the requested rate. The actual rate is returned in the variable. The actual rate should be used when configuring the UART IP. Note that 160MHz is used as that can be divided by the UART to hit all the common UART rates within required error. Other frequencies may have issues and the caller needs to calculate the baud clock error rate. See the UART section of the SoC RM.

```

/* Configure UART 0 baud clock */
sc_pm_set_clock_rate(ipc, SC_R_UART_0, SC_PM_CLK_PER, &rate);

```

Then enable the clock.

```

/* Enable UART 0 clock */
sc_pm_clock_enable(ipc, SC_R_UART_0, SC_PM_CLK_PER,      SC_TRUE, SC_FALSE);

```

At this point, the UART IP can be configured and used.

10.4.2 Macro Definition Documentation

10.4.2.1 SC_PM_CLK_MODE_ROM_INIT

```
#define SC_PM_CLK_MODE_ROM_INIT 0U
```

Clock is initialized by ROM.

10.4.2.2 SC_PM_CLK_MODE_ON

```
#define SC_PM_CLK_MODE_ON 2U
```

Clock is enabled.

10.4.2.3 SC_PM_PARENT_XTAL

```
#define SC_PM_PARENT_XTAL 0U
```

Parent is XTAL.

10.4.2.4 SC_PM_PARENT_BYPASS

```
#define SC_PM_PARENT_BYPASS 4U
```

Parent is a bypass clock.

10.4.3 Typedef Documentation

10.4.3.1 sc_pm_power_mode_t

```
typedef uint8_t sc_pm_power_mode_t
```

This type is used to declare a power mode.

Note resources only use SC_PM_PW_MODE_OFF and SC_PM_PW_MODE_ON. The other modes are used only as system power modes.

10.4.4 Function Documentation

10.4.4.1 sc_pm_set_sys_power_mode()

```
sc_err_t sc_pm_set_sys_power_mode (
    sc_ipc_t ipc,
    sc_pm_power_mode_t mode )
```

This function sets the system power mode.

Only the owner of the SC_R_SYSTEM resource can do this.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mode</i>	power mode to apply

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid mode,
- SC_ERR_NOACCESS if caller not the owner of SC_R_SYSTEM

See also

[sc_pm_set_sys_power_mode\(\)](#).

10.4.4.2 sc_pm_set_partition_power_mode()

```
sc_err_t sc_pm_set_partition_power_mode (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_pm_power_mode_t mode )
```

This function sets the power mode of a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition
in	<i>mode</i>	power mode to apply

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid partition or mode,
- SC_ERR_NOACCESS if caller's partition is not the owner or parent of *pt*

The power mode of the partitions is a max power any resource will be set to. Calling this will result in all resources owned by *pt* to have their power changed to the lower of *mode* or the individual resource mode set using [sc_pm_set_resource_power_mode\(\)](#).

10.4.4.3 sc_pm_get_sys_power_mode()

```
sc_err_t sc_pm_get_sys_power_mode (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_pm_power_mode_t * mode )
```

This function gets the power mode of a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition
out	<i>mode</i>	pointer to return power mode

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid partition

10.4.4.4 sc_pm_set_resource_power_mode()

```
sc_err_t sc_pm_set_resource_power_mode (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_power_mode_t mode )
```

This function sets the power mode of a resource.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>mode</i>	power mode to apply

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or mode,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner

Resources must be at SC_PM_PW_MODE_LP mode or higher to access them, otherwise the master will get a bus error or hang.

This function will record the individual resource power mode and change it if the requested mode is lower than or equal to the partition power mode set with [sc_pm_set_partition_power_mode\(\)](#). In other words, the power mode of the resource will be the minimum of the resource power mode and the partition power mode.

Note some resources are still not accessible even when powered up if bus transactions go through a fabric not powered up. Examples of this are resources in display and capture subsystems which require the display controller or the imaging subsystem to be powered up first.

Not that resources are grouped into power domains by the underlying hardware. If any resource in the domain is on, the entire power domain will be on. Other power domains required to access the resource will also be turned on. Clocks required to access the peripheral will be turned on. Refer to the SoC RM for more info on power domains and access infrastructure (bus fabrics, clock domains, etc.).

10.4.4.5 sc_pm_set_resource_power_mode_all()

```
sc_err_t sc_pm_set_resource_power_mode_all (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_pm_power_mode_t mode,
    sc_rsrc_t exclude )
```

This function sets the power mode for all the resources owned by a child partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of child partition
in	<i>mode</i>	power mode to apply
in	<i>exclude</i>	resource to exclude

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid partition or mode,
- SC_ERR_NOACCESS if caller's partition is not the parent of *pt*

This functions loops through all the resources owned by *pt* and sets the power mode to *mode*. It will skip setting *exclude* (SC_R_LAST to skip none).

This function can only be called by the parent. It is used to implement some aspects of virtualization.

10.4.4.6 sc_pm_get_resource_power_mode()

```
sc_err_t sc_pm_get_resource_power_mode (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_power_mode_t * mode )
```

This function gets the power mode of a resource.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
out	<i>mode</i>	pointer to return power mode

Returns

Returns an error code (SC_ERR_NONE = success).

Note only SC_PM_PW_MODE_OFF and SC_PM_PW_MODE_ON are valid. The value returned does not reflect the power mode of the partition..

10.4.4.7 sc_pm_req_low_power_mode()

```
sc_err_t sc_pm_req_low_power_mode (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_power_mode_t mode )
```

This function requests the low power mode some of the resources can enter based on their state.

This API is only valid for the following resources : SC_R_A53, SC_R_A53_0, SC_R_A53_1, SC_A53_2, SC_A53_3, SC_R_A72, SC_R_A72_0, SC_R_A72_1, SC_R_CC1, SC_R_A35, SC_R_A35_0, SC_R_A35_1, SC_R_A35_2, SC_R_A35_3. For all other resources it will return SC_ERR_PARAM. This function will set the low power mode the cores, cluster and cluster associated resources will enter when all the cores in a given cluster execute WFI

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>mode</i>	power mode to apply

Returns

Returns an error code (SC_ERR_NONE = success).

10.4.4.8 sc_pm_req_cpu_low_power_mode()

```
sc_err_t sc_pm_req_cpu_low_power_mode (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_power_mode_t mode,
    sc_pm_wake_src_t wake_src )
```

This function requests low-power mode entry for CPU/cluster resources.

This API is only valid for the following resources: SC_R_A53, SC_R_A53_x, SC_R_A72, SC_R_A72_x, SC_R_A35, SC_R_A35_x, SC_R_CCI. For all other resources it will return SC_ERR_PARAM. For individual core resources, the specified power mode and wake source will be applied after the core has entered WFI. For cluster resources, the specified power mode is applied after all cores in the cluster have entered low-power mode.

For multicluster resources, the specified power mode is applied after all clusters have reached low-power mode.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>mode</i>	power mode to apply
in	<i>wake_src</i>	wake source for low-power exit

Returns

Returns an error code (SC_ERR_NONE = success).

10.4.4.9 sc_pm_set_cpu_resume_addr()

```
sc_err_t sc_pm_set_cpu_resume_addr (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_faddr_t address )
```

This function is used to set the resume address of a CPU.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the CPU resource
in	<i>address</i>	64-bit resume address

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or address,
- SC_ERR_NOACCESS if caller's partition is not the parent of the resource (CPU) owner

10.4.4.10 sc_pm_set_cpu_resume()

```
sc_err_t sc_pm_set_cpu_resume (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_bool_t isPrimary,
    sc_faddr_t address )
```

This function is used to set parameters for CPU resume from low-power mode.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the CPU resource
in	<i>isPrimary</i>	set SC_TRUE if primary wake CPU
in	<i>address</i>	64-bit resume address

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or address,
- SC_ERR_NOACCESS if caller's partition is not the parent of the resource (CPU) owner

10.4.4.11 sc_pm_req_sys_if_power_mode()

```
sc_err_t sc_pm_req_sys_if_power_mode (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
```

```

sc_pm_sys_if_t sys_if,
sc_pm_power_mode_t hpm,
sc_pm_power_mode_t lpm )

```

This function requests the power mode configuration for system-level interfaces including messaging units, interconnect, and memories.

This API is only valid for the following resources : SC_R_A53, SC_R_A72, and SC_R_M4_x_PID_y. For all other resources, it will return SC_ERR_PARAM. The requested power mode will be captured and applied to system-level resources as system conditions allow.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>sys_if</i>	system-level interface to be configured
in	<i>hpm</i>	high-power mode for the system interface
in	<i>lpm</i>	low-power mode for the system interface

Returns

Returns an error code (SC_ERR_NONE = success).

10.4.4.12 sc_pm_set_clock_rate()

```

sc_err_t sc_pm_set_clock_rate (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_clk_t clk,
    sc_pm_clock_rate_t * rate )

```

This function sets the rate of a resource's clock/PLL.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>clk</i>	clock/PLL to affect
in, out	<i>rate</i>	pointer to rate to set, return actual rate

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or clock/PLL,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner,
- SC_ERR_UNAVAILABLE if clock/PLL not applicable to this resource,
- SC_ERR_LOCKED if rate locked (usually because shared clock/PLL)

Refer to the [Clock List](#) for valid clock/PLL values.

10.4.4.13 sc_pm_get_clock_rate()

```
sc_err_t sc_pm_get_clock_rate (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_clk_t clk,
    sc_pm_clock_rate_t * rate )
```

This function gets the rate of a resource's clock/PLL.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>clk</i>	clock/PLL to affect
out	<i>rate</i>	pointer to return rate

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or clock/PLL,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner,
- SC_ERR_UNAVAILABLE if clock/PLL not applicable to this resource

Refer to the [Clock List](#) for valid clock/PLL values.

10.4.4.14 sc_pm_clock_enable()

```
sc_err_t sc_pm_clock_enable (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_clk_t clk,
    sc_bool_t enable,
    sc_bool_t autog )
```

This function enables/disables a resource's clock.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>clk</i>	clock to affect
in	<i>enable</i>	enable if SC_TRUE; otherwise disabled
in	<i>autog</i>	HW auto clock gating

If *resource* is SC_R_ALL then all resources owned will be affected. No error will be returned.

If *clk* is SC_PM_CLK_ALL, then an error will be returned if any of the available clocks returns an error.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or clock,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner,
- SC_ERR_UNAVAILABLE if clock not applicable to this resource

Refer to the [Clock List](#) for valid clock values.

10.4.4.15 sc_pm_set_clock_parent()

```
sc_err_t sc_pm_set_clock_parent (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_clk_t clk,
    sc_pm_clk_parent_t parent )
```

This function sets the parent of a resource's clock.

This function should only be called when the clock is disabled.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>clk</i>	clock to affect
in	<i>parent</i>	New parent of the clock.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or clock,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner,
- SC_ERR_UNAVAILABLE if clock not applicable to this resource
- SC_ERR_BUSY if clock is currently enabled.
- SC_ERR_NOPOWER if resource not powered

Refer to the [Clock List](#) for valid clock values.

10.4.4.16 sc_pm_get_clock_parent()

```
sc_err_t sc_pm_get_clock_parent (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_pm_clk_t clk,
    sc_pm_clk_parent_t * parent )
```

This function gets the parent of a resource's clock.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the resource
in	<i>clk</i>	clock to affect
out	<i>parent</i>	pointer to return parent of clock.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or clock,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner,
- SC_ERR_UNAVAILABLE if clock not applicable to this resource

Refer to the [Clock List](#) for valid clock values.

10.4.4.17 sc_pm_reset()

```
sc_err_t sc_pm_reset (
    sc_ipc_t ipc,
    sc_pm_reset_type_t type )
```

This function is used to reset the system.

Only the owner of the SC_R_SYSTEM resource can do this.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>type</i>	reset type

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid type,
- SC_ERR_NOACCESS if caller not the owner of SC_R_SYSTEM

If this function returns, then the reset did not occur due to an invalid parameter.

10.4.4.18 sc_pm_reset_reason()

```
sc_err_t sc_pm_reset_reason (
    sc_ipc_t ipc,
    sc_pm_reset_reason_t * reason )
```

This function gets a caller's reset reason.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>reason</i>	pointer to return reset reason

This function returns the reason a partition was reset. If the reason is POR, then the system reset reason will be returned.

Note depending on the connection of the WDOG_OUT signal and the OTP programming of the PMIC, some reset reasons may trigger a system POR and the original reason will be lost.

Returns

Returns an error code (SC_ERR_NONE = success).

10.4.4.19 sc_pm_boot()

```
sc_err_t sc_pm_boot (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_rsrc_t resource_cpu,
    sc_faddr_t boot_addr,
    sc_rsrc_t resource_mu,
    sc_rsrc_t resource_dev )
```

This function is used to boot a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to boot
in	<i>resource_cpu</i>	ID of the CPU resource to start
in	<i>boot_addr</i>	64-bit boot address
in	<i>resource_mu</i>	ID of the MU that must be powered
in	<i>resource_dev</i>	ID of the boot device that must be powered

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid partition, resource, or addr,
- SC_ERR_NOACCESS if caller's partition is not the parent of the partition to boot

10.4.4.20 sc_pm_reboot()

```
void sc_pm_reboot (
    sc_ipc_t ipc,
    sc_pm_reset_type_t type )
```

This function is used to reboot the caller's partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>type</i>	reset type

If *type* is SC_PM_RESET_TYPE_COLD, then most peripherals owned by the calling partition will be reset if possible. SC state (partitions, power, clocks, etc.) is reset. The boot SW of the booting CPU must be able to handle peripherals that that are not reset.

If *type* is SC_PM_RESET_TYPE_WARM, then only the boot CPU is reset. SC state (partitions, power, clocks, etc.) are NOT reset. The boot SW of the booting CPU must be able to handle peripherals and SC state that that are not reset.

If *type* is SC_PM_RESET_TYPE_BOARD, then return with no action.

If this function returns, then the reset did not occur due to an invalid parameter.

10.4.4.21 sc_pm_reboot_partition()

```
sc_err_t sc_pm_reboot_partition (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_pm_reset_type_t type )
```

This function is used to reboot a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to reboot
in	<i>type</i>	reset type

If *type* is SC_PM_RESET_TYPE_COLD, then most peripherals owned by the calling partition will be reset if possible. SC state (partitions, power, clocks, etc.) is reset. The boot SW of the booting CPU must be able to handle peripherals that that are not reset.

If *type* is SC_PM_RESET_TYPE_WARM, then only the boot CPU is reset. SC state (partitions, power, clocks, etc.) are NOT reset. The boot SW of the booting CPU must be able to handle peripherals and SC state that that are not reset.

If *type* is SC_PM_RESET_TYPE_BOARD, then return with no action.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid partition or type
- SC_ERR_NOACCESS if caller's partition is not the parent of *pt*,

Most peripherals owned by the partition will be reset if possible. SC state (partitions, power, clocks, etc.) is reset. The boot SW of the booting CPU must be able to handle peripherals that that are not reset.

10.4.4.22 `sc_pm_cpu_start()`

```
sc_err_t sc_pm_cpu_start (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_bool_t enable,
    sc_faddr_t address )
```

This function is used to start/stop a CPU.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	ID of the CPU resource
in	<i>enable</i>	start if SC_TRUE; otherwise stop
in	<i>address</i>	64-bit boot address

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource or address,
- SC_ERR_NOACCESS if caller's partition is not the parent of the resource (CPU) owner

10.5 (SVC) Resource Management Service

Module for the Resource Management (RM) service.

Typedefs

- typedef `uint8_t sc_rm_pt_t`
This type is used to declare a resource partition.
- typedef `uint8_t sc_rm_mr_t`
This type is used to declare a memory region.
- typedef `uint8_t sc_rm_did_t`
This type is used to declare a resource domain ID used by the isolation HW.
- typedef `uint16_t sc_rm_sid_t`
This type is used to declare an SMMU StreamID.
- typedef `uint8_t sc_rm_spa_t`
This type is used to declare master transaction attributes.
- typedef `uint8_t sc_rm_perm_t`
This type is used to declare a resource/memory region access permission.

Defines for type widths

- #define `SC_RM_PARTITION_W` 5U
Width of `sc_rm_pt_t`.
- #define `SC_RM_MEMREG_W` 6U
Width of `sc_rm_mr_t`.
- #define `SC_RM_DID_W` 4U
Width of `sc_rm_did_t`.
- #define `SC_RM_SID_W` 6U
Width of `sc_rm_sid_t`.
- #define `SC_RM_SPA_W` 2U
Width of `sc_rm_spa_t`.
- #define `SC_RM_PERM_W` 3U
Width of `sc_rm_perm_t`.

Defines for ALL parameters

- #define `SC_RM_PT_ALL` ((`sc_rm_pt_t`) UINT8_MAX)
All partitions.
- #define `SC_RM_MR_ALL` ((`sc_rm_mr_t`) UINT8_MAX)
All memory regions.

Defines for `sc_rm_spa_t`

- `#define SC_RM_SPA_PASSTHRU 0U`
Pass through (attribute driven by master)
- `#define SC_RM_SPA_PASSSID 1U`
Pass through and output on SID.
- `#define SC_RM_SPA_ASSERT 2U`
Assert (force to be secure/privileged)
- `#define SC_RM_SPA_NEGATE 3U`
Negate (force to be non-secure/user)

Defines for `sc_rm_perm_t`

- `#define SC_RM_PERM_NONE 0U`
No access.
- `#define SC_RM_PERM_SEC_R 1U`
Secure RO.
- `#define SC_RM_PERM_SECPRIV_RW 2U`
Secure privilege R/W.
- `#define SC_RM_PERM_SEC_RW 3U`
Secure R/W.
- `#define SC_RM_PERM_NS_PRIV_R 4U`
Secure R/W, non-secure privilege RO.
- `#define SC_RM_PERM_NS_R 5U`
Secure R/W, non-secure RO.
- `#define SC_RM_PERM_NS_PRIV_RW 6U`
Secure R/W, non-secure privilege R/W.
- `#define SC_RM_PERM_FULL 7U`
Full access.

Partition Functions

- `sc_err_t sc_rm_partition_alloc` (`sc_ipc_t` ipc, `sc_rm_pt_t` *pt, `sc_bool_t` secure, `sc_bool_t` isolated, `sc_bool_t` restricted, `sc_bool_t` grant, `sc_bool_t` coherent)
This function requests that the SC create a new resource partition.
- `sc_err_t sc_rm_set_confidential` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_bool_t` retro)
This function makes a partition confidential.
- `sc_err_t sc_rm_partition_free` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt)
This function frees a partition and assigns all resources to the caller.
- `sc_rm_did_t sc_rm_get_did` (`sc_ipc_t` ipc)
This function returns the DID of a partition.
- `sc_err_t sc_rm_partition_static` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_rm_did_t` did)
This function forces a partition to use a specific static DID.
- `sc_err_t sc_rm_partition_lock` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt)
This function locks a partition.
- `sc_err_t sc_rm_get_partition` (`sc_ipc_t` ipc, `sc_rm_pt_t` *pt)

This function gets the partition handle of the caller.

- `sc_err_t sc_rm_set_parent` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_rm_pt_t` pt_parent)

This function sets a new parent for a partition.

- `sc_err_t sc_rm_move_all` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt_src, `sc_rm_pt_t` pt_dst, `sc_bool_t` move_rsrc, `sc_bool_t` move_pads)

This function moves all movable resources/pads owned by a source partition to a destination partition.

Resource Functions

- `sc_err_t sc_rm_assign_resource` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_rsrc_t` resource)

This function assigns ownership of a resource to a partition.

- `sc_err_t sc_rm_set_resource_movable` (`sc_ipc_t` ipc, `sc_rsrc_t` resource_fst, `sc_rsrc_t` resource_lst, `sc_bool_t` movable)

This function flags resources as movable or not.

- `sc_err_t sc_rm_set_subsys_rsrc_movable` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_bool_t` movable)

This function flags all of a subsystem's resources as movable or not.

- `sc_err_t sc_rm_set_master_attributes` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_rm_spa_t` sa, `sc_rm_spa_t` pa, `sc_bool_t` smmu_bypass)

This function sets attributes for a resource which is a bus master (i.e.

- `sc_err_t sc_rm_set_master_sid` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_rm_sid_t` sid)

This function sets the StreamID for a resource which is a bus master (i.e.

- `sc_err_t sc_rm_set_peripheral_permissions` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_rm_pt_t` pt, `sc_rm_perm_t` perm)

This function sets access permissions for a peripheral resource.

- `sc_bool_t sc_rm_is_resource_owned` (`sc_ipc_t` ipc, `sc_rsrc_t` resource)

This function gets ownership status of a resource.

- `sc_bool_t sc_rm_is_resource_master` (`sc_ipc_t` ipc, `sc_rsrc_t` resource)

This function is used to test if a resource is a bus master.

- `sc_bool_t sc_rm_is_resource_peripheral` (`sc_ipc_t` ipc, `sc_rsrc_t` resource)

This function is used to test if a resource is a peripheral.

- `sc_err_t sc_rm_get_resource_info` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_rm_sid_t` *sid)

This function is used to obtain info about a resource.

Memory Region Functions

- `sc_err_t sc_rm_memreg_alloc` (`sc_ipc_t` ipc, `sc_rm_mr_t` *mr, `sc_faddr_t` addr_start, `sc_faddr_t` addr_end)

This function requests that the SC create a new memory region.

- `sc_err_t sc_rm_memreg_split` (`sc_ipc_t` ipc, `sc_rm_mr_t` mr, `sc_rm_mr_t` *mr_ret, `sc_faddr_t` addr_start, `sc_faddr_t` addr_end)

This function requests that the SC split a memory region.

- `sc_err_t sc_rm_memreg_free` (`sc_ipc_t` ipc, `sc_rm_mr_t` mr)

This function frees a memory region.

- `sc_err_t sc_rm_find_memreg` (`sc_ipc_t` ipc, `sc_rm_mr_t` *mr, `sc_faddr_t` addr_start, `sc_faddr_t` addr_end)

Internal SC function to find a memory region.

- `sc_err_t sc_rm_assign_memreg` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_rm_mr_t` mr)

This function assigns ownership of a memory region.

- `sc_err_t sc_rm_set_memreg_permissions` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`, `sc_rm_pt_t pt`, `sc_rm_perm_t perm`)
This function sets access permissions for a memory region.
- `sc_bool_t sc_rm_is_memreg_owned` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`)
This function gets ownership status of a memory region.
- `sc_err_t sc_rm_get_memreg_info` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`, `sc_faddr_t *addr_start`, `sc_faddr_t *addr_end`)
This function is used to obtain info about a memory region.

Pad Functions

- `sc_err_t sc_rm_assign_pad` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_pad_t pad`)
This function assigns ownership of a pad to a partition.
- `sc_err_t sc_rm_set_pad_movable` (`sc_ipc_t ipc`, `sc_pad_t pad_fst`, `sc_pad_t pad_lst`, `sc_bool_t movable`)
This function flags pads as movable or not.
- `sc_bool_t sc_rm_is_pad_owned` (`sc_ipc_t ipc`, `sc_pad_t pad`)
This function gets ownership status of a pad.

Debug Functions

- `void sc_rm_dump` (`sc_ipc_t ipc`)
This function dumps the RM state for debug.

10.5.1 Detailed Description

Module for the Resource Management (RM) service.

The following SCFW resource manager (RM) code is an example of how to create a partition for an M4 core and its resources. This could be run from another core, an SCD, or be embedded into board.c.

The `ipc` parameter most functions take is a handle to the IPC channel opened to communicate to the SC. It is implementation defined. Most API ports include an `sc_ipc_open()` and `sc_ipc_close()` function to manage this. The `sc_ipc_open()` takes an argument to identify the communication channel (usually the MU address) and returns the IPC handle that all API calls should then use.

Note all this configuration can be done with the M4 subsystem powered off. It will be loaded when the M4 is powered on.

```
1 sc_rm_pt_t pt_m4_0;
2 sc_rm_mr_t mr_ddr1, mr_ddr2, mr_m4_0;
3
4 //sc_rm_dump(ipc);
5
6 /* Mark all resources as not movable */
7 sc_rm_set_resource_movable(ipc, SC_R_ALL, SC_R_ALL, SC_FALSE);
8 sc_rm_set_pad_movable(ipc, SC_P_ALL, SC_P_ALL, SC_FALSE);
9
10 /* Allocate M4_0 partition */
11 sc_rm_partition_alloc(ipc, &pt_m4_0, SC_FALSE, SC_TRUE, SC_FALSE, SC_TRUE,
12 SC_FALSE);
13
14 /* Mark all M4_0 subsystem resources as movable */
15 sc_rm_set_subsys_rsrc_movable(ipc, SC_R_M4_0_PID0, SC_TRUE);
16 sc_rm_set_pad_movable(ipc, SC_P_ADC_IN3, SC_P_ADC_IN2, SC_TRUE);
```

```

17
18 /* Keep some resources in the parent partition */
19 sc_rm_set_resource_movable(ipc, SC_R_M4_0_PID1, SC_R_M4_0_PID4,
20     SC_FALSE);
21 sc_rm_set_resource_movable(ipc, SC_R_M4_0_MU_0A0, SC_R_M4_0_MU_0A3,
22     SC_FALSE);
23
24 /* Move some resource not in the M4_0 subsystem */
25 sc_rm_set_resource_movable(ipc, SC_R_IRQSTR_M4_0, SC_R_IRQSTR_M4_0,
26     SC_TRUE);
27 sc_rm_set_resource_movable(ipc, SC_R_M4_1_MU_0A0, SC_R_M4_1_MU_0A0,
28     SC_TRUE);
29
30 /* Move everything flagged as movable */
31 sc_rm_move_all(ipc, ipc, pt_m4_0, SC_TRUE, SC_TRUE);
32
33 /* Allow all to access the SEMA42 */
34 sc_rm_set_peripheral_permissions(pt_m4_0, SC_R_M4_0_SEMA42,
35     SC_RM_PT_ALL, SC_RM_PERM_FULL);
36
37 /* Move M4_0 TCM */
38 sc_rm_find_memreg(ipc, &mr_m4_0, 0x034FE0000, 0x034FE0000);
39 sc_rm_assign_memreg(ipc, pt_m4_0, mr_m4_0);
40
41 /* Split DDR space, assign 0x88000000-0x8FFFFFFF to CM4 */
42 sc_rm_find_memreg(ipc, &mr_ddr1, 0x080000000, 0x080000000);
43 sc_rm_memreg_split(ipc, mr_ddr1, &mr_ddr2, 0x090000000, 0x0FFFFFFF);
44
45 /* Reserve DDR for M4_0 */
46 sc_rm_memreg_split(ipc, mr_ddr1, &mr_m4_0, 0x088000000, 0x08FFFFFFF);
47 sc_rm_assign_memreg(ipc, pt_m4_0, mr_m4_0);
48
49 //sc_rm_dump(ipc);

```

First, variables are declared to hold return partition and memory region handles.

```

sc_rm_pt_t pt_m4_0;
sc_rm_mr_t mr_ddr1, mr_ddr2, mr_m4_0;

```

Optionally, call `sc_rm_dump()` to dump the state of the RM to the SCFW debug UART.

```
//sc_rm_dump(ipc);
```

Mark resources and pins as movable or not movable to the new partition. By default, all resources are marked as movable. Marking all as movable or not movable first depends on how many resources are to be moved and which is the most efficient. Marking does not move the resource yet. Note, that it is also possible to assign resources individually using `sc_rm_assign_resource()`.

```

/* Mark all resources as not movable */
sc_rm_set_resource_movable(ipc, SC_R_ALL, SC_R_ALL, SC_FALSE);
sc_rm_set_pad_movable(ipc, SC_P_ALL, SC_P_ALL, SC_FALSE);

```

The `sc_rm_partition_alloc()` function call requests that the SC create a new partition to contain the M4 system. This function does not access the hardware at all. It allocates a new partition and returns a partition handle (`pt_m4_0`). The partition is marked non-secure as `secure=SC_FALSE`. Marking as non-secure prevents subsequent functions from configuring masters in this partition to assert the TZPROT signal.

```

/* Allocate M4_0 partition */
sc_rm_partition_alloc(ipc, &pt_m4_0, SC_FALSE, SC_TRUE, SC_FALSE, SC_TRUE,
    SC_FALSE);

```

Now mark some resources as movable. `sc_rm_set_subsys_rsrc_movable()` can be used to mark all resources in a HW subsystem. `sc_rm_set_pad_movable()` is used to mark some pads (i.e. pins) as movable.

```

/* Mark all M4_0 subsystem resources as movable */
sc_rm_set_subsys_rsrc_movable(ipc, SC_R_M4_0_PID0, SC_TRUE);
sc_rm_set_pad_movable(ipc, SC_P_ADC_IN3, SC_P_ADC_IN2, SC_TRUE);

```

Then mark some resources in the M4_0 subsystem (all marked movable above) as not movable using

`sc_rm_set_resource_movable()` In this case the process IDs used to access memory owned by other partitions as well as the MUs used for others to communicate with the M4 need to be left with the parent partition.

```
/* Keep some resources in the parent partition */
sc_rm_set_resource_movable(ipc, SC_R_M4_0_PID1, SC_R_M4_0_PID4,
    SC_FALSE);
sc_rm_set_resource_movable(ipc, SC_R_M4_0_MU_0A0, SC_R_M4_0_MU_0A3,
    SC_FALSE);
```

Move some resources in other subsystems. The new partition will require access to the IRQ Steer module which routes interrupts to this M4's NVIC. In this example, it also needs access to one of the M4_1 MUs.

```
/* Move some resource not in the M4_0 subsystem */
sc_rm_set_resource_movable(ipc, SC_R_IRQSTR_M4_0, SC_R_IRQSTR_M4_0,
```

Now assign (i.e. move) everything marked as movable. At this point, all these resources are in the new partition and HW will enforce isolation.

```
/* Move everything flagged as movable */
sc_rm_move_all(ipc, ipc, pt_m4_0, SC_TRUE, SC_TRUE);
```

Allow others to access some of the new partitions resources. In this case, the SEMA42 IP works by allowing multiple CPUs to access and acquire the semaphore.

```
/* Allow all to access the SEMA42 */
sc_rm_set_peripheral_permissions(pt_m4_0, SC_R_M4_0_SEMA42,
    SC_RM_PT_ALL, SC_RM_PERM_FULL);
```

Now assign the M4_0 TCM to the M4 partition. Note the M4 can always access its TCM. This action prevents the parent (current owner of the M4 TCM) from accessing. This should only be done after code for the M4 has been loaded into the TCM. Code loading will require the M4 subsystem already be powered on.

```
/* Move M4_0 TCM */
sc_rm_find_memreg(ipc, &mr_m4_0, 0x034FE0000, 0x034FE0000);
sc_rm_assign_memreg(ipc, pt_m4_0, mr_m4_0);
```

Next is to carve out some DDR for the M4. In this case, the memory is in the middle of the DDR so the DDR has to be split into three regions. First is to split off the end portion and keep this with the parent. Next is then to split off the end of the remaining part and assign this to the M4.

```
/* Split DDR space, assign 0x88000000-0x8FFFFFFF to CM4 */
sc_rm_find_memreg(ipc, &mr_ddr1, 0x080000000, 0x080000000);
sc_rm_memreg_split(ipc, mr_ddr1, &mr_ddr2, 0x090000000, 0x0FFFFFFF);
/* Reserve DDR for M4_0 */
sc_rm_memreg_split(ipc, mr_ddr1, &mr_m4_0, 0x088000000, 0x08FFFFFFF);
sc_rm_assign_memreg(ipc, pt_m4_0, mr_m4_0);
```

Optionally, call `sc_rm_dump()` to dump the state of the RM to the SCFW debug UART.

```
//sc_rm_dump(ipc);
```

At this point, the M4 can be powered on (if not already) and the M4 can be started using `sc_pm_boot()`. *Do NOT start the CPU using `sc_pm_cpu_start()` as that function is for starting a secondary CPU in the calling core's partition.* In this case, the core is in another partition that needs to be booted.

Refer to the SoC-specific [Resource List](#) for a list of resources.

10.5.2 Typedef Documentation

10.5.2.1 `sc_rm_perm_t`

```
typedef uint8_t sc_rm_perm_t
```

This type is used to declare a resource/memory region access permission.

Refer to the XRDC2 Block Guide for more information.

10.5.3 Function Documentation

10.5.3.1 `sc_rm_partition_alloc()`

```
sc_err_t sc_rm_partition_alloc (
    sc_ipc_t ipc,
    sc_rm_pt_t * pt,
    sc_bool_t secure,
    sc_bool_t isolated,
    sc_bool_t restricted,
    sc_bool_t grant,
    sc_bool_t coherent )
```

This function requests that the SC create a new resource partition.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>pt</i>	return handle for partition; used for subsequent function calls associated with this partition
in	<i>secure</i>	boolean indicating if this partition should be secure; only valid if caller is secure
in	<i>isolated</i>	boolean indicating if this partition should be HW isolated via XRDC; set SC_TRUE if new DID is desired
in	<i>restricted</i>	boolean indicating if this partition should be restricted; set SC_TRUE if masters in this partition cannot create new partitions
in	<i>grant</i>	boolean indicating if this partition should always grant access and control to the parent
in	<i>coherent</i>	boolean indicating if this partition is coherent; set SC_TRUE if only this partition will contain both AP clusters and they will be coherent via the CCI

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,

- SC_ERR_PARM if caller's partition is not secure but a new secure partition is requested,
- SC_ERR_LOCKED if caller's partition is locked,
- SC_ERR_UNAVAILABLE if partition table is full (no more allocation space)

Marking as non-secure prevents subsequent functions from configuring masters in this partition to assert the secure signal. If restricted then the new partition is limited in what functions it can call, especially those associated with managing partitions.

The grant option is usually used to isolate a bus master's traffic to specific memory without isolating the peripheral interface of the master or the API controls of that master.

10.5.3.2 `sc_rm_set_confidential()`

```
sc_err_t sc_rm_set_confidential (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_bool_t retro )
```

This function makes a partition confidential.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition that is granting
in	<i>retro</i>	retroactive

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if *pt* out of range,
- SC_ERR_NOACCESS if caller's not allowed to change *pt*
- SC_ERR_LOCKED if partition *pt* is locked

Call to make a partition confidential. Confidential means only this partition should be able to grant access permissions to this partition.

If retroactive, then all resources owned by other partitions will have access rights for this partition removed, even if locked.

10.5.3.3 `sc_rm_partition_free()`

```
sc_err_t sc_rm_partition_free (
    sc_ipc_t ipc,
    sc_rm_pt_t pt )
```

This function frees a partition and assigns all resources to the caller.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to free

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,
- SC_PARM if *pt* out of range or invalid,
- SC_ERR_NOACCESS if *pt* is the SC partition,
- SC_ERR_NOACCESS if caller's partition is not the parent of *pt*,
- SC_ERR_LOCKED if *pt* or caller's partition is locked

All resources, memory regions, and pads are assigned to the caller/parent. The partition watchdog is disabled (even if locked). DID is freed.

10.5.3.4 sc_rm_get_did()

```
sc_rm_did_t sc_rm_get_did (  
    sc_ipc_t ipc )
```

This function returns the DID of a partition.

Parameters

in	<i>ipc</i>	IPC handle
----	------------	------------

Returns

Returns the domain ID (DID) of the caller's partition.

The DID is a SoC-specific internal ID used by the HW resource protection mechanism. It is only required by clients when using the SEMA42 module as the DID is sometimes connected to the master ID.

10.5.3.5 sc_rm_partition_static()

```
sc_err_t sc_rm_partition_static (  
    sc_ipc_t ipc,
```



```
sc_rm_pt_t pt,  
sc_rm_did_t did )
```

This function forces a partition to use a specific static DID.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to assign <i>did</i>
in	<i>did</i>	static DID to assign

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,
- SC_PARM if *pt* or *did* out of range,
- SC_ERR_NOACCESS if caller's partition is not the parent of *pt*,
- SC_ERR_LOCKED if *pt* is locked

Assumes no assigned resources or memory regions yet! The number of static DID is fixed by the SC at boot.

10.5.3.6 sc_rm_partition_lock()

```
sc_err_t sc_rm_partition_lock (
    sc_ipc_t ipc,
    sc_rm_pt_t pt )
```

This function locks a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to lock

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if *pt* out of range,
- SC_ERR_NOACCESS if caller's partition is not the parent of *pt*

If a partition is locked it cannot be freed, have resources/pads assigned to/from it, memory regions created/assigned, DID changed, or parent changed.

10.5.3.7 sc_rm_get_partition()

```
sc_err_t sc_rm_get_partition (
    sc_ipc_t ipc,
    sc_rm_pt_t * pt )
```

This function gets the partition handle of the caller.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>pt</i>	return handle for caller's partition

Returns

Returns an error code (SC_ERR_NONE = success).

10.5.3.8 sc_rm_set_parent()

```
sc_err_t sc_rm_set_parent (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_rm_pt_t pt_parent )
```

This function sets a new parent for a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition for which parent is to be changed
in	<i>pt_parent</i>	handle of partition to set as parent

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,
- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the parent of *pt*,
- SC_ERR_LOCKED if either partition is locked

10.5.3.9 sc_rm_move_all()

```
sc_err_t sc_rm_move_all (
    sc_ipc_t ipc,
    sc_rm_pt_t pt_src,
    sc_rm_pt_t pt_dst,
    sc_bool_t move_rsrc,
    sc_bool_t move_pads )
```

This function moves all movable resources/pads owned by a source partition to a destination partition.

It can be used to more quickly set up a new partition if a majority of the caller's resources are to be moved to a new partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt_src</i>	handle of partition from which resources should be moved from
in	<i>pt_dst</i>	handle of partition to which resources should be moved to
in	<i>move_rsrc</i>	boolean to indicate if resources should be moved
in	<i>move_pads</i>	boolean to indicate if pads should be moved

Returns

Returns an error code (SC_ERR_NONE = success).

By default, all resources are movable. This can be changed using the [sc_rm_set_resource_movable\(\)](#) function. Note all masters defaulted to SMMU bypass.

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,
- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not *pt_src* or the parent of *pt_src*,
- SC_ERR_LOCKED if either partition is locked

10.5.3.10 sc_rm_assign_resource()

```
sc_err_t sc_rm_assign_resource (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_rsrc_t resource )
```

This function assigns ownership of a resource to a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to which resource should be assigned
in	<i>resource</i>	resource to assign

Returns

Returns an error code (SC_ERR_NONE = success).

This action resets the resource's master and peripheral attributes. Privilege attribute will be PASSTHRU, security attribute will be ASSERT if the partition is secure and NEGATE if it is not, and masters will default to SMMU bypass. Access permissions will reset to SEC_RW for the owning partition only for secure partitions, FULL for non-secure. Default is no access by other partitions.

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,
- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner,
- SC_ERR_LOCKED if the owning partition or *pt* is locked

10.5.3.11 sc_rm_set_resource_movable()

```
sc_err_t sc_rm_set_resource_movable (
    sc_ipc_t ipc,
    sc_rsrc_t resource_fst,
    sc_rsrc_t resource_lst,
    sc_bool_t movable )
```

This function flags resources as movable or not.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource_fst</i>	first resource for which flag should be set
in	<i>resource_lst</i>	last resource for which flag should be set
in	<i>movable</i>	movable flag (SC_TRUE is movable)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if resources are out of range,
- SC_ERR_NOACCESS if caller's partition is not a parent of a resource owner,
- SC_ERR_LOCKED if the owning partition is locked

This function is used to determine the set of resources that will be moved using the [sc_rm_move_all\(\)](#) function. All resources are movable by default so this function is normally used to prevent a set of resources from moving.

10.5.3.12 sc_rm_set_subsys_rsrc_movable()

```
sc_err_t sc_rm_set_subsys_rsrc_movable (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_bool_t movable )
```

This function flags all of a subsystem's resources as movable or not.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource to use to identify subsystem
in	<i>movable</i>	movable flag (SC_TRUE is movable)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if a function argument is out of range

Note *resource* is used to find the associated subsystem. Only resources owned by the caller are set.

10.5.3.13 sc_rm_set_master_attributes()

```
sc_err_t sc_rm_set_master_attributes (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_rm_spa_t sa,
    sc_rm_spa_t pa,
    sc_bool_t smmu_bypass )
```

This function sets attributes for a resource which is a bus master (i.e. capable of DMA).

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	master resource for which attributes should apply
in	<i>sa</i>	security attribute
in	<i>pa</i>	privilege attribute
in	<i>smmu_bypass</i>	SMMU bypass mode

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,
- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not a parent of the resource owner,
- SC_ERR_LOCKED if the owning partition is locked

This function configures how the HW isolation will see bus transactions from the specified master. Note the security attribute will only be changed if the caller's partition is secure.

10.5.3.14 sc_rm_set_master_sid()

```
sc_err_t sc_rm_set_master_sid (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_rm_sid_t sid )
```

This function sets the StreamID for a resource which is a bus master (i.e. capable of DMA).

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	master resource for which attributes should apply
in	<i>sid</i>	StreamID

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,
- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner,
- SC_ERR_LOCKED if the owning partition is locked

This function configures the SID attribute associated with all bus transactions from this master. Note 0 is not a valid SID as it is reserved to indicate bypass.

10.5.3.15 `sc_rm_set_peripheral_permissions()`

```
sc_err_t sc_rm_set_peripheral_permissions (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_rm_pt_t pt,
    sc_rm_perm_t perm )
```

This function sets access permissions for a peripheral resource.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	peripheral resource for which permissions should apply
in	<i>pt</i>	handle of partition <i>perm</i> should be applied for
in	<i>perm</i>	permissions to apply to <i>resource</i> for <i>pt</i>

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent of the owner,
- SC_ERR_LOCKED if the owning partition is locked
- SC_ERR_LOCKED if the *pt* is confidential and the caller isn't *pt*

This function configures how the HW isolation will restrict access to a peripheral based on the attributes of a transaction from bus master.

10.5.3.16 `sc_rm_is_resource_owned()`

```
sc_bool_t sc_rm_is_resource_owned (
    sc_ipc_t ipc,
    sc_rsrc_t resource )
```

This function gets ownership status of a resource.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource to check

Returns

Returns a boolean (SC_TRUE if caller's partition owns the resource).

If *resource* is out of range then SC_FALSE is returned.

10.5.3.17 sc_rm_is_resource_master()

```
sc_bool_t sc_rm_is_resource_master (
    sc_ipc_t ipc,
    sc_rsrc_t resource )
```

This function is used to test if a resource is a bus master.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource to check

Returns

Returns a boolean (SC_TRUE if the resource is a bus master).

If *resource* is out of range then SC_FALSE is returned.

10.5.3.18 sc_rm_is_resource_peripheral()

```
sc_bool_t sc_rm_is_resource_peripheral (
    sc_ipc_t ipc,
    sc_rsrc_t resource )
```

This function is used to test if a resource is a peripheral.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource to check

Returns

Returns a boolean (SC_TRUE if the resource is a peripheral).

If *resource* is out of range then SC_FALSE is returned.

10.5.3.19 sc_rm_get_resource_info()

```
sc_err_t sc_rm_get_resource_info (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_rm_sid_t * sid )
```

This function is used to obtain info about a resource.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource to inquire about
out	<i>sid</i>	pointer to return StreamID

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if *resource* is out of range

10.5.3.20 sc_rm_memreg_alloc()

```
sc_err_t sc_rm_memreg_alloc (
    sc_ipc_t ipc,
    sc_rm_mr_t * mr,
    sc_faddr_t addr_start,
    sc_faddr_t addr_end )
```

This function requests that the SC create a new memory region.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>mr</i>	return handle for region; used for subsequent function calls associated with this region
in	<i>addr_start</i>	start address of region (physical)
in	<i>addr_end</i>	end address of region (physical)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if the new memory region is misaligned,
- SC_ERR_LOCKED if caller's partition is locked,
- SC_ERR_PARM if the new memory region spans multiple existing regions,
- SC_ERR_NOACCESS if caller's partition does not own the memory containing the new region,
- SC_ERR_UNAVAILABLE if memory region table is full (no more allocation space)

The area covered by the memory region must currently be owned by the caller. By default, the new region will have access permission set to allow the caller to access.

10.5.3.21 sc_rm_memreg_split()

```
sc_err_t sc_rm_memreg_split (
    sc_ipc_t ipc,
    sc_rm_mr_t mr,
    sc_rm_mr_t * mr_ret,
    sc_faddr_t addr_start,
    sc_faddr_t addr_end )
```

This function requests that the SC split a memory region.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mr</i>	handle of memory region to split
out	<i>mr_ret</i>	return handle for new region; used for subsequent function calls associated with this region
in	<i>addr_start</i>	start address of region (physical)
in	<i>addr_end</i>	end address of region (physical)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if the new memory region is not start/end part of mr,
- SC_ERR_LOCKED if caller's partition is locked,
- SC_ERR_PARM if the new memory region spans multiple existing regions,

- SC_ERR_NOACCESS if caller's partition does not own the memory containing the new region,
- SC_ERR_UNAVAILABLE if memory region table is full (no more allocation space)

Note the new region must start or end on the split region.

10.5.3.22 `sc_rm_memreg_free()`

```
sc_err_t sc_rm_memreg_free (
    sc_ipc_t ipc,
    sc_rm_mr_t mr )
```

This function frees a memory region.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mr</i>	handle of memory region to free

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if *mr* out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not a parent of *mr*,
- SC_ERR_LOCKED if the owning partition of *mr* is locked

10.5.3.23 `sc_rm_find_memreg()`

```
sc_err_t sc_rm_find_memreg (
    sc_ipc_t ipc,
    sc_rm_mr_t * mr,
    sc_faddr_t addr_start,
    sc_faddr_t addr_end )
```

Internal SC function to find a memory region.

See also

[sc_rm_find_memreg\(\)](#).

This function finds a memory region.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>mr</i>	return handle for region; used for subsequent function calls associated with this region
in	<i>addr_start</i>	start address of region to search for
in	<i>addr_end</i>	end address of region to search for

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_NOTFOUND if region not found,

Searches only for regions owned by the caller. Finds first region containing the range specified.

10.5.3.24 sc_rm_assign_memreg()

```
sc_err_t sc_rm_assign_memreg (  
    sc_ipc_t ipc,  
    sc_rm_pt_t pt,  
    sc_rm_mr_t mr )
```

This function assigns ownership of a memory region.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to which memory region should be assigned
in	<i>mr</i>	handle of memory region to assign

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the *mr* owner or parent of the owner,
- SC_ERR_LOCKED if the owning partition or *pt* is locked

10.5.3.25 sc_rm_set_memreg_permissions()

```
sc_err_t sc_rm_set_memreg_permissions (
    sc_ipc_t ipc,
    sc_rm_mr_t mr,
    sc_rm_pt_t pt,
    sc_rm_perm_t perm )
```

This function sets access permissions for a memory region.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mr</i>	handle of memory region for which permissions should apply
in	<i>pt</i>	handle of partition <i>perm</i> should be applied for
in	<i>perm</i>	permissions to apply to <i>mr</i> for <i>pt</i>

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the region owner or parent of the owner,
- SC_ERR_LOCKED if the owning partition is locked
- SC_ERR_LOCKED if the *pt* is confidential and the caller isn't *pt*

This function configures how the HW isolation will restrict access to a memory region based on the attributes of a transaction from bus master.

10.5.3.26 sc_rm_is_memreg_owned()

```
sc_bool_t sc_rm_is_memreg_owned (
    sc_ipc_t ipc,
    sc_rm_mr_t mr )
```

This function gets ownership status of a memory region.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mr</i>	handle of memory region to check

Returns

Returns a boolean (SC_TRUE if caller's partition owns the memory region).

If *mr* is out of range then SC_FALSE is returned.

10.5.3.27 sc_rm_get_memreg_info()

```
sc_err_t sc_rm_get_memreg_info (
    sc_ipc_t ipc,
    sc_rm_mr_t mr,
    sc_faddr_t * addr_start,
    sc_faddr_t * addr_end )
```

This function is used to obtain info about a memory region.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mr</i>	handle of memory region to inquire about
out	<i>addr_start</i>	pointer to return start address
out	<i>addr_end</i>	pointer to return end address

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if *mr* is out of range

10.5.3.28 sc_rm_assign_pad()

```
sc_err_t sc_rm_assign_pad (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_pad_t pad )
```

This function assigns ownership of a pad to a partition.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	handle of partition to which pad should be assigned
in	<i>pad</i>	pad to assign

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_NOACCESS if caller's partition is restricted,
- SC_PARM if arguments out of range or invalid,
- SC_ERR_NOACCESS if caller's partition is not the pad owner or parent of the owner,
- SC_ERR_LOCKED if the owning partition or *pt* is locked

10.5.3.29 sc_rm_set_pad_movable()

```
sc_err_t sc_rm_set_pad_movable (
    sc_ipc_t ipc,
    sc_pad_t pad_fst,
    sc_pad_t pad_lst,
    sc_bool_t movable )
```

This function flags pads as movable or not.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad_fst</i>	first pad for which flag should be set
in	<i>pad_lst</i>	last pad for which flag should be set
in	<i>movable</i>	movable flag (SC_TRUE is movable)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_PARM if pads are out of range,
- SC_ERR_NOACCESS if caller's partition is not a parent of a pad owner,
- SC_ERR_LOCKED if the owning partition is locked

This function is used to determine the set of pads that will be moved using the [sc_rm_move_all\(\)](#) function. All pads are movable by default so this function is normally used to prevent a set of pads from moving.

10.5.3.30 sc_rm_is_pad_owned()

```
sc_bool_t sc_rm_is_pad_owned (
    sc_ipc_t ipc,
    sc_pad_t pad )
```

This function gets ownership status of a pad.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pad</i>	pad to check

Returns

Returns a boolean (SC_TRUE if caller's partition owns the pad).

If *pad* is out of range then SC_FALSE is returned.

10.5.3.31 sc_rm_dump()

```
void sc_rm_dump (
    sc_ipc_t ipc )
```

This function dumps the RM state for debug.

Parameters

in	<i>ipc</i>	IPC handle
----	------------	------------

10.6 (SVC) Timer Service

Module for the Timer service.

Typedefs

- typedef `uint8_t sc_timer_wdog_action_t`
This type is used to configure the watchdog action.
- typedef `uint32_t sc_timer_wdog_time_t`
This type is used to declare a watchdog time value in milliseconds.

Defines for type widths

- #define `SC_TIMER_ACTION_W` 3U
Width of `sc_timer_wdog_action_t`.

Defines for `sc_timer_wdog_action_t`

- #define `SC_TIMER_WDOG_ACTION_PARTITION` 0U
Reset partition.
- #define `SC_TIMER_WDOG_ACTION_WARM` 1U
Warm reset system.
- #define `SC_TIMER_WDOG_ACTION_COLD` 2U
Cold reset system.
- #define `SC_TIMER_WDOG_ACTION_BOARD` 3U
Reset board.
- #define `SC_TIMER_WDOG_ACTION_IRQ` 4U
Only generate IRQs.

Wathdog Functions

- `sc_err_t sc_timer_set_wdog_timeout` (`sc_ipc_t ipc`, `sc_timer_wdog_time_t timeout`)
This function sets the watchdog timeout in milliseconds.
- `sc_err_t sc_timer_set_wdog_pre_timeout` (`sc_ipc_t ipc`, `sc_timer_wdog_time_t pre_timeout`)
This function sets the watchdog pre-timeout in milliseconds.
- `sc_err_t sc_timer_start_wdog` (`sc_ipc_t ipc`, `sc_bool_t lock`)
This function starts the watchdog.
- `sc_err_t sc_timer_stop_wdog` (`sc_ipc_t ipc`)
This function stops the watchdog if it is not locked.
- `sc_err_t sc_timer_ping_wdog` (`sc_ipc_t ipc`)
This function pings (services, kicks) the watchdog resetting the time before expiration back to the timeout.
- `sc_err_t sc_timer_get_wdog_status` (`sc_ipc_t ipc`, `sc_timer_wdog_time_t *timeout`, `sc_timer_wdog_time_t *max_timeout`, `sc_timer_wdog_time_t *remaining_time`)
This function gets the status of the watchdog.
- `sc_err_t sc_timer_pt_get_wdog_status` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_bool_t *enb`, `sc_timer_wdog_time_t *timeout`, `sc_timer_wdog_time_t *remaining_time`)
This function gets the status of the watchdog of a partition.
- `sc_err_t sc_timer_set_wdog_action` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_timer_wdog_action_t action`)
This function configures the action to be taken when a watchdog expires.

Real-Time Clock (RTC) Functions

- `sc_err_t sc_timer_set_rtc_time` (`sc_ipc_t ipc`, `uint16_t year`, `uint8_t mon`, `uint8_t day`, `uint8_t hour`, `uint8_t min`, `uint8_t sec`)
This function sets the RTC time.
- `sc_err_t sc_timer_get_rtc_time` (`sc_ipc_t ipc`, `uint16_t *year`, `uint8_t *mon`, `uint8_t *day`, `uint8_t *hour`, `uint8_t *min`, `uint8_t *sec`)
This function gets the RTC time.
- `sc_err_t sc_timer_get_rtc_sec1970` (`sc_ipc_t ipc`, `uint32_t *sec`)
This function gets the RTC time in seconds since 1/1/1970.
- `sc_err_t sc_timer_set_rtc_alarm` (`sc_ipc_t ipc`, `uint16_t year`, `uint8_t mon`, `uint8_t day`, `uint8_t hour`, `uint8_t min`, `uint8_t sec`)
This function sets the RTC alarm.
- `sc_err_t sc_timer_set_rtc_periodic_alarm` (`sc_ipc_t ipc`, `uint32_t sec`)
This function sets the RTC alarm (periodic mode).
- `sc_err_t sc_timer_cancel_rtc_alarm` (`sc_ipc_t ipc`)
This function cancels the RTC alarm.
- `sc_err_t sc_timer_set_rtc_calb` (`sc_ipc_t ipc`, `int8_t count`)
This function sets the RTC calibration value.

System Counter (SYSCTR) Functions

- `sc_err_t sc_timer_set_sysctr_alarm` (`sc_ipc_t ipc`, `uint64_t ticks`)
This function sets the SYSCTR alarm.
- `sc_err_t sc_timer_set_sysctr_periodic_alarm` (`sc_ipc_t ipc`, `uint64_t ticks`)
This function sets the SYSCTR alarm (periodic mode).
- `sc_err_t sc_timer_cancel_sysctr_alarm` (`sc_ipc_t ipc`)
This function cancels the SYSCTR alarm.

10.6.1 Detailed Description

Module for the Timer service.

This includes support for the watchdog, RTC, and system counter. Note every resource partition has a watchdog it can use.

10.6.2 Function Documentation

10.6.2.1 `sc_timer_set_wdog_timeout()`

```
sc_err_t sc_timer_set_wdog_timeout (
    sc_ipc_t ipc,
    sc_timer_wdog_time_t timeout )
```

This function sets the watchdog timeout in milliseconds.

If not set then the timeout defaults to the max. Once locked this value cannot be changed.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>timeout</i>	timeout period for the watchdog

Returns

Returns an error code (SC_ERR_NONE = success, SC_ERR_LOCKED = locked).

10.6.2.2 sc_timer_set_wdog_pre_timeout()

```
sc_err_t sc_timer_set_wdog_pre_timeout (
    sc_ipc_t ipc,
    sc_timer_wdog_time_t pre_timeout )
```

This function sets the watchdog pre-timeout in milliseconds.

If not set then the pre-timeout defaults to the max. Once locked this value cannot be changed.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pre_timeout</i>	pre-timeout period for the watchdog

When the pre-timeout expires an IRQ will be generated. Note this timeout clears when the IRQ is triggered. An IRQ is generated for the failing partition and all of its child partitions.

Returns

Returns an error code (SC_ERR_NONE = success).

10.6.2.3 sc_timer_start_wdog()

```
sc_err_t sc_timer_start_wdog (
    sc_ipc_t ipc,
    sc_bool_t lock )
```

This function starts the watchdog.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>lock</i>	boolean indicating the lock status

Returns

Returns an error code (SC_ERR_NONE = success).

If *lock* is set then the watchdog cannot be stopped or the timeout period changed.

10.6.2.4 sc_timer_stop_wdog()

```
sc_err_t sc_timer_stop_wdog (  
    sc_ipc_t ipc )
```

This function stops the watchdog if it is not locked.

Parameters

in	<i>ipc</i>	IPC handle
----	------------	------------

Returns

Returns an error code (SC_ERR_NONE = success, SC_ERR_LOCKED = locked).

10.6.2.5 sc_timer_ping_wdog()

```
sc_err_t sc_timer_ping_wdog (  
    sc_ipc_t ipc )
```

This function pings (services, kicks) the watchdog resetting the time before expiration back to the timeout.

Parameters

in	<i>ipc</i>	IPC handle
----	------------	------------

Returns

Returns an error code (SC_ERR_NONE = success).

10.6.2.6 sc_timer_get_wdog_status()

```
sc_err_t sc_timer_get_wdog_status (
    sc_ipc_t ipc,
    sc_timer_wdog_time_t * timeout,
    sc_timer_wdog_time_t * max_timeout,
    sc_timer_wdog_time_t * remaining_time )
```

This function gets the status of the watchdog.

All arguments are in milliseconds.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>timeout</i>	pointer to return the timeout
out	<i>max_timeout</i>	pointer to return the max timeout
out	<i>remaining_time</i>	pointer to return the time remaining until trigger

Returns

Returns an error code (SC_ERR_NONE = success).

10.6.2.7 sc_timer_pt_get_wdog_status()

```
sc_err_t sc_timer_pt_get_wdog_status (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_bool_t * enb,
    sc_timer_wdog_time_t * timeout,
    sc_timer_wdog_time_t * remaining_time )
```

This function gets the status of the watchdog of a partition.

All arguments are in milliseconds.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	partition to query
out	<i>enb</i>	pointer to return enable status
out	<i>timeout</i>	pointer to return the timeout
out	<i>remaining_time</i>	pointer to return the time remaining until trigger

Returns

Returns an error code (SC_ERR_NONE = success).

10.6.2.8 sc_timer_set_wdog_action()

```
sc_err_t sc_timer_set_wdog_action (
    sc_ipc_t ipc,
    sc_rm_pt_t pt,
    sc_timer_wdog_action_t action )
```

This function configures the action to be taken when a watchdog expires.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>pt</i>	partition to affect
in	<i>action</i>	action to take

Default action is inherited from the parent.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid parameters,
- SC_ERR_NOACCESS if caller's partition is not the SYSTEM owner,
- SC_ERR_LOCKED if the watchdog is locked

10.6.2.9 sc_timer_set_rtc_time()

```
sc_err_t sc_timer_set_rtc_time (
    sc_ipc_t ipc,
    uint16_t year,
    uint8_t mon,
    uint8_t day,
    uint8_t hour,
    uint8_t min,
    uint8_t sec )
```

This function sets the RTC time.

Only the owner of the SC_R_SYSTEM resource can set the time.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>year</i>	year (min 1970)
in	<i>mon</i>	month (1-12)
in	<i>day</i>	day of the month (1-31)
in	<i>hour</i>	hour (0-23)
in	<i>min</i>	minute (0-59)
in	<i>sec</i>	second (0-59)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters,
- SC_ERR_NOACCESS if caller's partition is not the SYSTEM owner

10.6.2.10 sc_timer_get_rtc_time()

```
sc_err_t sc_timer_get_rtc_time (
    sc_ipc_t ipc,
    uint16_t * year,
    uint8_t * mon,
    uint8_t * day,
    uint8_t * hour,
    uint8_t * min,
    uint8_t * sec )
```

This function gets the RTC time.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>year</i>	pointer to return year (min 1970)
out	<i>mon</i>	pointer to return month (1-12)
out	<i>day</i>	pointer to return day of the month (1-31)
out	<i>hour</i>	pointer to return hour (0-23)
out	<i>min</i>	pointer to return minute (0-59)
out	<i>sec</i>	pointer to return second (0-59)

Returns

Returns an error code (SC_ERR_NONE = success).

10.6.2.11 sc_timer_get_rtc_sec1970()

```
sc_err_t sc_timer_get_rtc_sec1970 (
    sc_ipc_t ipc,
    uint32_t * sec )
```

This function gets the RTC time in seconds since 1/1/1970.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>sec</i>	pointer to return second

Returns

Returns an error code (SC_ERR_NONE = success).

10.6.2.12 sc_timer_set_rtc_alarm()

```
sc_err_t sc_timer_set_rtc_alarm (
    sc_ipc_t ipc,
    uint16_t year,
    uint8_t mon,
    uint8_t day,
    uint8_t hour,
    uint8_t min,
    uint8_t sec )
```

This function sets the RTC alarm.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>year</i>	year (min 1970)
in	<i>mon</i>	month (1-12)
in	<i>day</i>	day of the month (1-31)
in	<i>hour</i>	hour (0-23)
in	<i>min</i>	minute (0-59)
in	<i>sec</i>	second (0-59)

Note this alarm setting clears when the alarm is triggered.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters

10.6.2.13 sc_timer_set_rtc_periodic_alarm()

```
sc_err_t sc_timer_set_rtc_periodic_alarm (
    sc_ipc_t ipc,
    uint32_t sec )
```

This function sets the RTC alarm (periodic mode).

Parameters

in	<i>ipc</i>	IPC handle
in	<i>sec</i>	period in seconds

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters

10.6.2.14 sc_timer_cancel_rtc_alarm()

```
sc_err_t sc_timer_cancel_rtc_alarm (
    sc_ipc_t ipc )
```

This function cancels the RTC alarm.

Parameters

<i>in</i>	<i>ipc</i>	IPC handle
-----------	------------	------------

Note this alarm setting clears when the alarm is triggered.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters

10.6.2.15 sc_timer_set_rtc_calb()

```
sc_err_t sc_timer_set_rtc_calb (  
    sc_ipc_t ipc,  
    int8_t count )
```

This function sets the RTC calibration value.

Only the owner of the SC_R_SYSTEM resource can set the calibration.

Parameters

<i>in</i>	<i>ipc</i>	IPC handle
<i>in</i>	<i>count</i>	calbration count (-16 to 15)

The calibration value is a 5-bit value including the sign bit, which is implemented in 2's complement. It is added or subtracted from the RTC on a peridodic basis, once per 32768 cycles of the RTC clock.

Returns

Returns an error code (SC_ERR_NONE = success).

10.6.2.16 sc_timer_set_sysctr_alarm()

```
sc_err_t sc_timer_set_sysctr_alarm (  
    sc_ipc_t ipc,  
    uint64_t ticks )
```

This function sets the SYSCTR alarm.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>ticks</i>	number of 8MHz cycles

Note this alarm setting clears when the alarm is triggered.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters

10.6.2.17 sc_timer_set_sysctr_periodic_alarm()

```
sc_err_t sc_timer_set_sysctr_periodic_alarm (
    sc_ipc_t ipc,
    uint64_t ticks )
```

This function sets the SYSCTR alarm (periodic mode).

Parameters

in	<i>ipc</i>	IPC handle
in	<i>ticks</i>	number of 8MHz cycles

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters

10.6.2.18 sc_timer_cancel_sysctr_alarm()

```
sc_err_t sc_timer_cancel_sysctr_alarm (
    sc_ipc_t ipc )
```

This function cancels the SYSCTR alarm.

Parameters

in	<i>ipc</i>	IPC handle
----	------------	------------

Note this alarm setting clears when the alarm is triggered.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters

Chapter 11

File Documentation

11.1 platform/main/ipc.h File Reference

Header file for the IPC implementation.

Functions

- `sc_err_t sc_ipc_open` (`sc_ipc_t *ipc`, `sc_ipc_id_t id`)
This function opens an IPC channel.
- `void sc_ipc_close` (`sc_ipc_t ipc`)
This function closes an IPC channel.
- `void sc_ipc_read` (`sc_ipc_t ipc`, `void *data`)
This function reads a message from an IPC channel.
- `void sc_ipc_write` (`sc_ipc_t ipc`, `const void *data`)
This function writes a message to an IPC channel.

11.1.1 Detailed Description

Header file for the IPC implementation.

11.1.2 Function Documentation

11.1.2.1 `sc_ipc_open()`

```
sc_err_t sc_ipc_open (  
    sc_ipc_t * ipc,  
    sc_ipc_id_t id )
```

This function opens an IPC channel.

Parameters

out	<i>ipc</i>	return pointer for ipc handle
in	<i>id</i>	id of channel to open

Returns

Returns an error code (SC_ERR_NONE = success, SC_ERR_IPC otherwise).

The *id* parameter is implementation specific. Could be an MU address, pointer to a driver path, channel index, etc.

11.1.2.2 sc_ipc_close()

```
void sc_ipc_close (
    sc_ipc_t ipc )
```

This function closes an IPC channel.

Parameters

in	<i>ipc</i>	id of channel to close
----	------------	------------------------

11.1.2.3 sc_ipc_read()

```
void sc_ipc_read (
    sc_ipc_t ipc,
    void * data )
```

This function reads a message from an IPC channel.

Parameters

in	<i>ipc</i>	id of channel read from
out	<i>data</i>	pointer to message buffer to read

This function will block if no message is available to be read.

11.1.2.4 sc_ipc_write()

```
void sc_ipc_write (
    sc_ipc_t ipc,
    const void * data )
```


This function writes a message to an IPC channel.

Parameters

in	<i>ipc</i>	id of channel to write to
in	<i>data</i>	pointer to message buffer to write

This function will block if the outgoing buffer is full.

11.2 platform/main/types.h File Reference

Header file containing types used across multiple service APIs.

Macros

- `#define SC_C_TEMP 0U`
Defines for sc_ctrl_t.
- `#define SC_C_TEMP_HI 1U`
- `#define SC_C_TEMP_LOW 2U`
- `#define SC_C_PXL_LINK_MST1_ADDR 3U`
- `#define SC_C_PXL_LINK_MST2_ADDR 4U`
- `#define SC_C_PXL_LINK_MST_ENB 5U`
- `#define SC_C_PXL_LINK_MST1_ENB 6U`
- `#define SC_C_PXL_LINK_MST2_ENB 7U`
- `#define SC_C_PXL_LINK_SLV1_ADDR 8U`
- `#define SC_C_PXL_LINK_SLV2_ADDR 9U`
- `#define SC_C_PXL_LINK_MST_VLD 10U`
- `#define SC_C_PXL_LINK_MST1_VLD 11U`
- `#define SC_C_PXL_LINK_MST2_VLD 12U`
- `#define SC_C_SINGLE_MODE 13U`
- `#define SC_C_ID 14U`
- `#define SC_C_PXL_CLK_POLARITY 15U`
- `#define SC_C_LINESTATE 16U`
- `#define SC_C_PCIE_G_RST 17U`
- `#define SC_C_PCIE_BUTTON_RST 18U`
- `#define SC_C_PCIE_PERST 19U`
- `#define SC_C_PHY_RESET 20U`
- `#define SC_C_PXL_LINK_RATE_CORRECTION 21U`
- `#define SC_C_PANIC 22U`
- `#define SC_C_PRIORITY_GROUP 23U`
- `#define SC_C_TXCLK 24U`
- `#define SC_C_CLKDIV 25U`
- `#define SC_C_DISABLE_50 26U`
- `#define SC_C_DISABLE_125 27U`
- `#define SC_C_SEL_125 28U`
- `#define SC_C_MODE 29U`
- `#define SC_C_SYNC_CTRL0 30U`
- `#define SC_C_KACHUNK_CNT 31U`

- #define **SC_C_KACHUNK_SEL** 32U
- #define **SC_C_SYNC_CTRL1** 33U
- #define **SC_C_DPI_RESET** 34U
- #define **SC_C_MIPI_RESET** 35U
- #define **SC_C_DUAL_MODE** 36U
- #define **SC_C_VOLTAGE** 37U
- #define **SC_C_PXL_LINK_SEL** 38U
- #define **SC_C_OFS_SEL** 39U
- #define **SC_C_OFS_AUDIO** 40U
- #define **SC_C_OFS_PERIPH** 41U
- #define **SC_C_OFS_IRQ** 42U
- #define **SC_C_RST0** 43U
- #define **SC_C_RST1** 44U
- #define **SC_C_SEL0** 45U
- #define **SC_C_CALIB0** 46U
- #define **SC_C_CALIB1** 47U
- #define **SC_C_CALIB2** 48U
- #define **SC_C_IPG_DEBUG** 49U
- #define **SC_C_IPG_DOZE** 50U
- #define **SC_C_IPG_WAIT** 51U
- #define **SC_C_IPG_STOP** 52U
- #define **SC_C_IPG_STOP_MODE** 53U
- #define **SC_C_IPG_STOP_ACK** 54U
- #define **SC_C_SYNC_CTRL** 55U
- #define **SC_C_LAST** 56U
- #define **SC_P_ALL** ((**sc_pad_t**) UINT16_MAX)

All pads.

Defines for common frequencies

- #define **SC_32KHZ** 32768U
32KHz
- #define **SC_10MHZ** 10000000U
10MHz
- #define **SC_20MHZ** 20000000U
20MHz
- #define **SC_25MHZ** 25000000U
25MHz
- #define **SC_27MHZ** 27000000U
27MHz
- #define **SC_40MHZ** 40000000U
40MHz
- #define **SC_45MHZ** 45000000U
45MHz
- #define **SC_50MHZ** 50000000U
50MHz
- #define **SC_60MHZ** 60000000U
60MHz
- #define **SC_66MHZ** 66666666U
66MHz
- #define **SC_74MHZ** 74250000U
74.25MHz

- #define SC_80MHZ 80000000U
80MHz
- #define SC_83MHZ 83333333U
83MHz
- #define SC_84MHZ 84375000U
84.37MHz
- #define SC_100MHZ 100000000U
100MHz
- #define SC_125MHZ 125000000U
125MHz
- #define SC_133MHZ 133333333U
133MHz
- #define SC_135MHZ 135000000U
135MHz
- #define SC_150MHZ 150000000U
150MHz
- #define SC_160MHZ 160000000U
160MHz
- #define SC_166MHZ 166666666U
166MHz
- #define SC_175MHZ 175000000U
175MHz
- #define SC_180MHZ 180000000U
180MHz
- #define SC_200MHZ 200000000U
200MHz
- #define SC_250MHZ 250000000U
250MHz
- #define SC_266MHZ 266666666U
266MHz
- #define SC_300MHZ 300000000U
300MHz
- #define SC_312MHZ 312500000U
312.5MHz
- #define SC_320MHZ 320000000U
320MHz
- #define SC_325MHZ 325000000U
325MHz
- #define SC_333MHZ 333333333U
333MHz
- #define SC_350MHZ 350000000U
350MHz
- #define SC_372MHZ 372000000U
372MHz
- #define SC_375MHZ 375000000U
375MHz
- #define SC_400MHZ 400000000U
400MHz
- #define SC_500MHZ 500000000U
500MHz
- #define SC_594MHZ 594000000U
594MHz
- #define SC_625MHZ 625000000U
625MHz
- #define SC_640MHZ 640000000U

- 640MHz
 - #define SC_648MHZ 648000000U
- 648MHz
 - #define SC_650MHZ 650000000U
- 650MHz
 - #define SC_667MHZ 666666667U
- 667MHz
 - #define SC_675MHZ 675000000U
- 675MHz
 - #define SC_700MHZ 700000000U
- 700MHz
 - #define SC_720MHZ 720000000U
- 720MHz
 - #define SC_750MHZ 750000000U
- 750MHz
 - #define SC_753MHZ 753000000U
- 753MHz
 - #define SC_793MHZ 793000000U
- 793MHz
 - #define SC_800MHZ 800000000U
- 800MHz
 - #define SC_850MHZ 850000000U
- 850MHz
 - #define SC_858MHZ 858000000U
- 858MHz
 - #define SC_900MHZ 900000000U
- 900MHz
 - #define SC_953MHZ 953000000U
- 953MHz
 - #define SC_963MHZ 963000000U
- 963MHz
 - #define SC_1000MHZ 1000000000U
- 1GHz
 - #define SC_1060MHZ 1060000000U
- 1.06GHz
 - #define SC_1068MHZ 1068000000U
- 1.068GHz
 - #define SC_1121MHZ 1121000000U
- 1.121GHz
 - #define SC_1173MHZ 1173000000U
- 1.173GHz
 - #define SC_1188MHZ 1188000000U
- 1.188GHz
 - #define SC_1260MHZ 1260000000U
- 1.26GHz
 - #define SC_1278MHZ 1278000000U
- 1.278GHz
 - #define SC_1280MHZ 1280000000U
- 1.28GHz
 - #define SC_1300MHZ 1300000000U
- 1.3GHz
 - #define SC_1313MHZ 1313000000U
- 1.313GHz
 - #define SC_1345MHZ 1345000000U
- 1.345GHz

- #define SC_1400MHZ 1400000000U
1.4GHz
- #define SC_1500MHZ 1500000000U
1.5GHz
- #define SC_1600MHZ 1600000000U
1.6GHz
- #define SC_1800MHZ 1800000000U
1.8GHz
- #define SC_2000MHZ 2000000000U
2.0GHz
- #define SC_2112MHZ 2112000000U
2.12GHz

Defines for 24M related frequencies

- #define SC_8MHZ 8000000U
8MHz
- #define SC_12MHZ 12000000U
12MHz
- #define SC_19MHZ 19800000U
19.8MHz
- #define SC_24MHZ 24000000U
24MHz
- #define SC_48MHZ 48000000U
48MHz
- #define SC_120MHZ 120000000U
120MHz
- #define SC_132MHZ 132000000U
132MHz
- #define SC_144MHZ 144000000U
144MHz
- #define SC_192MHZ 192000000U
192MHz
- #define SC_211MHZ 211200000U
211.2MHz
- #define SC_240MHZ 240000000U
240MHz
- #define SC_264MHZ 264000000U
264MHz
- #define SC_352MHZ 352000000U
352MHz
- #define SC_360MHZ 360000000U
360MHz
- #define SC_384MHZ 384000000U
384MHz
- #define SC_396MHZ 396000000U
396MHz
- #define SC_432MHZ 432000000U
432MHz
- #define SC_480MHZ 480000000U
480MHz
- #define SC_600MHZ 600000000U
600MHz
- #define SC_744MHZ 744000000U

- 744MHz
- #define [SC_792MHZ](#) 792000000U
- 792MHz
- #define [SC_864MHZ](#) 864000000U
- 864MHz
- #define [SC_960MHZ](#) 960000000U
- 960MHz
- #define [SC_1056MHZ](#) 1056000000U
- 1056MHz
- #define [SC_1104MHZ](#) 1104000000U
- 1104MHz
- #define [SC_1200MHZ](#) 1200000000U
- 1.2GHz
- #define [SC_1464MHZ](#) 1464000000U
- 1.464GHz
- #define [SC_2400MHZ](#) 2400000000U
- 2.4GHz

Defines for A/V related frequencies

- #define [SC_62MHZ](#) 62937500U
- 62.9375MHz
- #define [SC_755MHZ](#) 755250000U
- 755.25MHz

Defines for type widths

- #define [SC_FADDR_W](#) 36U
- Width of [sc_faddr_t](#).*
- #define [SC_BOOL_W](#) 1U
- Width of [sc_bool_t](#).*
- #define [SC_ERR_W](#) 4U
- Width of [sc_err_t](#).*
- #define [SC_RSRC_W](#) 10U
- Width of [sc_rsrc_t](#).*
- #define [SC_CTRL_W](#) 6U
- Width of [sc_ctrl_t](#).*

Defines for [sc_bool_t](#)

- #define [SC_FALSE](#) (([sc_bool_t](#)) 0U)
- False.*
- #define [SC_TRUE](#) (([sc_bool_t](#)) 1U)
- True.*

Defines for [sc_err_t](#).

- #define [SC_ERR_NONE](#) 0U
- Success.*
- #define [SC_ERR_VERSION](#) 1U
- Incompatible API version.*
- #define [SC_ERR_CONFIG](#) 2U

- *Configuration error.*
- #define SC_ERR_PARM 3U
- *Bad parameter.*
- #define SC_ERR_NOACCESS 4U
- *Permission error (no access)*
- #define SC_ERR_LOCKED 5U
- *Permission error (locked)*
- #define SC_ERR_UNAVAILABLE 6U
- *Unavailable (out of resources)*
- #define SC_ERR_NOTFOUND 7U
- *Not found.*
- #define SC_ERR_NOPOWER 8U
- *No power.*
- #define SC_ERR_IPC 9U
- *Generic IPC error.*
- #define SC_ERR_BUSY 10U
- *Resource is currently busy/active.*
- #define SC_ERR_FAIL 11U
- *General I/O failure.*
- #define SC_ERR_LAST 12U

Defines for sc_rsrc_t.

- #define SC_R_A53 0U
- #define SC_R_A53_0 1U
- #define SC_R_A53_1 2U
- #define SC_R_A53_2 3U
- #define SC_R_A53_3 4U
- #define SC_R_A72 5U
- #define SC_R_A72_0 6U
- #define SC_R_A72_1 7U
- #define SC_R_A72_2 8U
- #define SC_R_A72_3 9U
- #define SC_R_CCI 10U
- #define SC_R_DB 11U
- #define SC_R_DRC_0 12U
- #define SC_R_DRC_1 13U
- #define SC_R_GIC_SMMU 14U
- #define SC_R_IRQSTR_M4_0 15U
- #define SC_R_IRQSTR_M4_1 16U
- #define SC_R_SMMU 17U
- #define SC_R_GIC 18U
- #define SC_R_DC_0_BLIT0 19U
- #define SC_R_DC_0_BLIT1 20U
- #define SC_R_DC_0_BLIT2 21U
- #define SC_R_DC_0_BLIT_OUT 22U
- #define SC_R_DC_0_CAPTURE0 23U
- #define SC_R_DC_0_CAPTURE1 24U
- #define SC_R_DC_0_WARP 25U
- #define SC_R_DC_0_INTEGRAL0 26U
- #define SC_R_DC_0_INTEGRAL1 27U
- #define SC_R_DC_0_VIDEO0 28U
- #define SC_R_DC_0_VIDEO1 29U
- #define SC_R_DC_0_FRAC0 30U
- #define SC_R_DC_0_FRAC1 31U
- #define SC_R_DC_0 32U
- #define SC_R_GPU_2_PID0 33U

- #define SC_R_DC_0_PLL_0 34U
- #define SC_R_DC_0_PLL_1 35U
- #define SC_R_DC_1_BLIT0 36U
- #define SC_R_DC_1_BLIT1 37U
- #define SC_R_DC_1_BLIT2 38U
- #define SC_R_DC_1_BLIT_OUT 39U
- #define SC_R_DC_1_CAPTURE0 40U
- #define SC_R_DC_1_CAPTURE1 41U
- #define SC_R_DC_1_WARP 42U
- #define SC_R_DC_1_INTEGRAL0 43U
- #define SC_R_DC_1_INTEGRAL1 44U
- #define SC_R_DC_1_VIDEO0 45U
- #define SC_R_DC_1_VIDEO1 46U
- #define SC_R_DC_1_FRAC0 47U
- #define SC_R_DC_1_FRAC1 48U
- #define SC_R_DC_1 49U
- #define SC_R_GPU_3_PID0 50U
- #define SC_R_DC_1_PLL_0 51U
- #define SC_R_DC_1_PLL_1 52U
- #define SC_R_SPI_0 53U
- #define SC_R_SPI_1 54U
- #define SC_R_SPI_2 55U
- #define SC_R_SPI_3 56U
- #define SC_R_UART_0 57U
- #define SC_R_UART_1 58U
- #define SC_R_UART_2 59U
- #define SC_R_UART_3 60U
- #define SC_R_UART_4 61U
- #define SC_R_EMVSIM_0 62U
- #define SC_R_EMVSIM_1 63U
- #define SC_R_DMA_0_CH0 64U
- #define SC_R_DMA_0_CH1 65U
- #define SC_R_DMA_0_CH2 66U
- #define SC_R_DMA_0_CH3 67U
- #define SC_R_DMA_0_CH4 68U
- #define SC_R_DMA_0_CH5 69U
- #define SC_R_DMA_0_CH6 70U
- #define SC_R_DMA_0_CH7 71U
- #define SC_R_DMA_0_CH8 72U
- #define SC_R_DMA_0_CH9 73U
- #define SC_R_DMA_0_CH10 74U
- #define SC_R_DMA_0_CH11 75U
- #define SC_R_DMA_0_CH12 76U
- #define SC_R_DMA_0_CH13 77U
- #define SC_R_DMA_0_CH14 78U
- #define SC_R_DMA_0_CH15 79U
- #define SC_R_DMA_0_CH16 80U
- #define SC_R_DMA_0_CH17 81U
- #define SC_R_DMA_0_CH18 82U
- #define SC_R_DMA_0_CH19 83U
- #define SC_R_DMA_0_CH20 84U
- #define SC_R_DMA_0_CH21 85U
- #define SC_R_DMA_0_CH22 86U
- #define SC_R_DMA_0_CH23 87U
- #define SC_R_DMA_0_CH24 88U
- #define SC_R_DMA_0_CH25 89U
- #define SC_R_DMA_0_CH26 90U
- #define SC_R_DMA_0_CH27 91U
- #define SC_R_DMA_0_CH28 92U
- #define SC_R_DMA_0_CH29 93U

- #define **SC_R_DMA_0_CH30** 94U
- #define **SC_R_DMA_0_CH31** 95U
- #define **SC_R_I2C_0** 96U
- #define **SC_R_I2C_1** 97U
- #define **SC_R_I2C_2** 98U
- #define **SC_R_I2C_3** 99U
- #define **SC_R_I2C_4** 100U
- #define **SC_R_ADC_0** 101U
- #define **SC_R_ADC_1** 102U
- #define **SC_R_FTM_0** 103U
- #define **SC_R_FTM_1** 104U
- #define **SC_R_CAN_0** 105U
- #define **SC_R_CAN_1** 106U
- #define **SC_R_CAN_2** 107U
- #define **SC_R_DMA_1_CH0** 108U
- #define **SC_R_DMA_1_CH1** 109U
- #define **SC_R_DMA_1_CH2** 110U
- #define **SC_R_DMA_1_CH3** 111U
- #define **SC_R_DMA_1_CH4** 112U
- #define **SC_R_DMA_1_CH5** 113U
- #define **SC_R_DMA_1_CH6** 114U
- #define **SC_R_DMA_1_CH7** 115U
- #define **SC_R_DMA_1_CH8** 116U
- #define **SC_R_DMA_1_CH9** 117U
- #define **SC_R_DMA_1_CH10** 118U
- #define **SC_R_DMA_1_CH11** 119U
- #define **SC_R_DMA_1_CH12** 120U
- #define **SC_R_DMA_1_CH13** 121U
- #define **SC_R_DMA_1_CH14** 122U
- #define **SC_R_DMA_1_CH15** 123U
- #define **SC_R_DMA_1_CH16** 124U
- #define **SC_R_DMA_1_CH17** 125U
- #define **SC_R_DMA_1_CH18** 126U
- #define **SC_R_DMA_1_CH19** 127U
- #define **SC_R_DMA_1_CH20** 128U
- #define **SC_R_DMA_1_CH21** 129U
- #define **SC_R_DMA_1_CH22** 130U
- #define **SC_R_DMA_1_CH23** 131U
- #define **SC_R_DMA_1_CH24** 132U
- #define **SC_R_DMA_1_CH25** 133U
- #define **SC_R_DMA_1_CH26** 134U
- #define **SC_R_DMA_1_CH27** 135U
- #define **SC_R_DMA_1_CH28** 136U
- #define **SC_R_DMA_1_CH29** 137U
- #define **SC_R_DMA_1_CH30** 138U
- #define **SC_R_DMA_1_CH31** 139U
- #define **SC_R_UNUSED1** 140U
- #define **SC_R_UNUSED2** 141U
- #define **SC_R_UNUSED3** 142U
- #define **SC_R_UNUSED4** 143U
- #define **SC_R_GPU_0_PID0** 144U
- #define **SC_R_GPU_0_PID1** 145U
- #define **SC_R_GPU_0_PID2** 146U
- #define **SC_R_GPU_0_PID3** 147U
- #define **SC_R_GPU_1_PID0** 148U
- #define **SC_R_GPU_1_PID1** 149U
- #define **SC_R_GPU_1_PID2** 150U
- #define **SC_R_GPU_1_PID3** 151U
- #define **SC_R_PCIE_A** 152U
- #define **SC_R_SERDES_0** 153U

- `#define SC_R_MATCH_0` 154U
- `#define SC_R_MATCH_1` 155U
- `#define SC_R_MATCH_2` 156U
- `#define SC_R_MATCH_3` 157U
- `#define SC_R_MATCH_4` 158U
- `#define SC_R_MATCH_5` 159U
- `#define SC_R_MATCH_6` 160U
- `#define SC_R_MATCH_7` 161U
- `#define SC_R_MATCH_8` 162U
- `#define SC_R_MATCH_9` 163U
- `#define SC_R_MATCH_10` 164U
- `#define SC_R_MATCH_11` 165U
- `#define SC_R_MATCH_12` 166U
- `#define SC_R_MATCH_13` 167U
- `#define SC_R_MATCH_14` 168U
- `#define SC_R_PCIE_B` 169U
- `#define SC_R_SATA_0` 170U
- `#define SC_R_SERDES_1` 171U
- `#define SC_R_HSIO_GPIO` 172U
- `#define SC_R_MATCH_15` 173U
- `#define SC_R_MATCH_16` 174U
- `#define SC_R_MATCH_17` 175U
- `#define SC_R_MATCH_18` 176U
- `#define SC_R_MATCH_19` 177U
- `#define SC_R_MATCH_20` 178U
- `#define SC_R_MATCH_21` 179U
- `#define SC_R_MATCH_22` 180U
- `#define SC_R_MATCH_23` 181U
- `#define SC_R_MATCH_24` 182U
- `#define SC_R_MATCH_25` 183U
- `#define SC_R_MATCH_26` 184U
- `#define SC_R_MATCH_27` 185U
- `#define SC_R_MATCH_28` 186U
- `#define SC_R_LCD_0` 187U
- `#define SC_R_LCD_0_PWM_0` 188U
- `#define SC_R_LCD_0_I2C_0` 189U
- `#define SC_R_LCD_0_I2C_1` 190U
- `#define SC_R_PWM_0` 191U
- `#define SC_R_PWM_1` 192U
- `#define SC_R_PWM_2` 193U
- `#define SC_R_PWM_3` 194U
- `#define SC_R_PWM_4` 195U
- `#define SC_R_PWM_5` 196U
- `#define SC_R_PWM_6` 197U
- `#define SC_R_PWM_7` 198U
- `#define SC_R_GPIO_0` 199U
- `#define SC_R_GPIO_1` 200U
- `#define SC_R_GPIO_2` 201U
- `#define SC_R_GPIO_3` 202U
- `#define SC_R_GPIO_4` 203U
- `#define SC_R_GPIO_5` 204U
- `#define SC_R_GPIO_6` 205U
- `#define SC_R_GPIO_7` 206U
- `#define SC_R_GPT_0` 207U
- `#define SC_R_GPT_1` 208U
- `#define SC_R_GPT_2` 209U
- `#define SC_R_GPT_3` 210U
- `#define SC_R_GPT_4` 211U
- `#define SC_R_KPP` 212U
- `#define SC_R_MU_0A` 213U

- #define SC_R_MU_1A 214U
- #define SC_R_MU_2A 215U
- #define SC_R_MU_3A 216U
- #define SC_R_MU_4A 217U
- #define SC_R_MU_5A 218U
- #define SC_R_MU_6A 219U
- #define SC_R_MU_7A 220U
- #define SC_R_MU_8A 221U
- #define SC_R_MU_9A 222U
- #define SC_R_MU_10A 223U
- #define SC_R_MU_11A 224U
- #define SC_R_MU_12A 225U
- #define SC_R_MU_13A 226U
- #define SC_R_MU_5B 227U
- #define SC_R_MU_6B 228U
- #define SC_R_MU_7B 229U
- #define SC_R_MU_8B 230U
- #define SC_R_MU_9B 231U
- #define SC_R_MU_10B 232U
- #define SC_R_MU_11B 233U
- #define SC_R_MU_12B 234U
- #define SC_R_MU_13B 235U
- #define SC_R_ROM_0 236U
- #define SC_R_FSPI_0 237U
- #define SC_R_FSPI_1 238U
- #define SC_R_IEE 239U
- #define SC_R_IEE_R0 240U
- #define SC_R_IEE_R1 241U
- #define SC_R_IEE_R2 242U
- #define SC_R_IEE_R3 243U
- #define SC_R_IEE_R4 244U
- #define SC_R_IEE_R5 245U
- #define SC_R_IEE_R6 246U
- #define SC_R_IEE_R7 247U
- #define SC_R_SDHC_0 248U
- #define SC_R_SDHC_1 249U
- #define SC_R_SDHC_2 250U
- #define SC_R_ENET_0 251U
- #define SC_R_ENET_1 252U
- #define SC_R_MLB_0 253U
- #define SC_R_DMA_2_CH0 254U
- #define SC_R_DMA_2_CH1 255U
- #define SC_R_DMA_2_CH2 256U
- #define SC_R_DMA_2_CH3 257U
- #define SC_R_DMA_2_CH4 258U
- #define SC_R_USB_0 259U
- #define SC_R_USB_1 260U
- #define SC_R_USB_0_PHY 261U
- #define SC_R_USB_2 262U
- #define SC_R_USB_2_PHY 263U
- #define SC_R_DTCP 264U
- #define SC_R_NAND 265U
- #define SC_R_LVDS_0 266U
- #define SC_R_LVDS_0_PWM_0 267U
- #define SC_R_LVDS_0_I2C_0 268U
- #define SC_R_LVDS_0_I2C_1 269U
- #define SC_R_LVDS_1 270U
- #define SC_R_LVDS_1_PWM_0 271U
- #define SC_R_LVDS_1_I2C_0 272U
- #define SC_R_LVDS_1_I2C_1 273U

- #define **SC_R_LVDS_2** 274U
- #define **SC_R_LVDS_2_PWM_0** 275U
- #define **SC_R_LVDS_2_I2C_0** 276U
- #define **SC_R_LVDS_2_I2C_1** 277U
- #define **SC_R_M4_0_PID0** 278U
- #define **SC_R_M4_0_PID1** 279U
- #define **SC_R_M4_0_PID2** 280U
- #define **SC_R_M4_0_PID3** 281U
- #define **SC_R_M4_0_PID4** 282U
- #define **SC_R_M4_0_RGPIO** 283U
- #define **SC_R_M4_0_SEMA42** 284U
- #define **SC_R_M4_0_TPM** 285U
- #define **SC_R_M4_0_PIT** 286U
- #define **SC_R_M4_0_UART** 287U
- #define **SC_R_M4_0_I2C** 288U
- #define **SC_R_M4_0_INTMUX** 289U
- #define **SC_R_M4_0_SIM** 290U
- #define **SC_R_M4_0_WDOG** 291U
- #define **SC_R_M4_0_MU_0B** 292U
- #define **SC_R_M4_0_MU_0A0** 293U
- #define **SC_R_M4_0_MU_0A1** 294U
- #define **SC_R_M4_0_MU_0A2** 295U
- #define **SC_R_M4_0_MU_0A3** 296U
- #define **SC_R_M4_0_MU_1A** 297U
- #define **SC_R_M4_1_PID0** 298U
- #define **SC_R_M4_1_PID1** 299U
- #define **SC_R_M4_1_PID2** 300U
- #define **SC_R_M4_1_PID3** 301U
- #define **SC_R_M4_1_PID4** 302U
- #define **SC_R_M4_1_RGPIO** 303U
- #define **SC_R_M4_1_SEMA42** 304U
- #define **SC_R_M4_1_TPM** 305U
- #define **SC_R_M4_1_PIT** 306U
- #define **SC_R_M4_1_UART** 307U
- #define **SC_R_M4_1_I2C** 308U
- #define **SC_R_M4_1_INTMUX** 309U
- #define **SC_R_M4_1_SIM** 310U
- #define **SC_R_M4_1_WDOG** 311U
- #define **SC_R_M4_1_MU_0B** 312U
- #define **SC_R_M4_1_MU_0A0** 313U
- #define **SC_R_M4_1_MU_0A1** 314U
- #define **SC_R_M4_1_MU_0A2** 315U
- #define **SC_R_M4_1_MU_0A3** 316U
- #define **SC_R_M4_1_MU_1A** 317U
- #define **SC_R_SAI_0** 318U
- #define **SC_R_SAI_1** 319U
- #define **SC_R_SAI_2** 320U
- #define **SC_R_IRQSTR_SCU2** 321U
- #define **SC_R_IRQSTR_DSP** 322U
- #define **SC_R_ELCDIF_PLL** 323U
- #define **SC_R_OCRAM** 324U
- #define **SC_R_AUDIO_PLL_0** 325U
- #define **SC_R_PI_0** 326U
- #define **SC_R_PI_0_PWM_0** 327U
- #define **SC_R_PI_0_PWM_1** 328U
- #define **SC_R_PI_0_I2C_0** 329U
- #define **SC_R_PI_0_PLL** 330U
- #define **SC_R_PI_1** 331U
- #define **SC_R_PI_1_PWM_0** 332U
- #define **SC_R_PI_1_PWM_1** 333U

- #define SC_R_PI_1_I2C_0 334U
- #define SC_R_PI_1_PLL 335U
- #define SC_R_SC_PID0 336U
- #define SC_R_SC_PID1 337U
- #define SC_R_SC_PID2 338U
- #define SC_R_SC_PID3 339U
- #define SC_R_SC_PID4 340U
- #define SC_R_SC_SEMA42 341U
- #define SC_R_SC_TPM 342U
- #define SC_R_SC_PIT 343U
- #define SC_R_SC_UART 344U
- #define SC_R_SC_I2C 345U
- #define SC_R_SC_MU_0B 346U
- #define SC_R_SC_MU_0A0 347U
- #define SC_R_SC_MU_0A1 348U
- #define SC_R_SC_MU_0A2 349U
- #define SC_R_SC_MU_0A3 350U
- #define SC_R_SC_MU_1A 351U
- #define SC_R_SYSCNT_RD 352U
- #define SC_R_SYSCNT_CMP 353U
- #define SC_R_DEBUG 354U
- #define SC_R_SYSTEM 355U
- #define SC_R_SNVS 356U
- #define SC_R_OTP 357U
- #define SC_R_VPU_PID0 358U
- #define SC_R_VPU_PID1 359U
- #define SC_R_VPU_PID2 360U
- #define SC_R_VPU_PID3 361U
- #define SC_R_VPU_PID4 362U
- #define SC_R_VPU_PID5 363U
- #define SC_R_VPU_PID6 364U
- #define SC_R_VPU_PID7 365U
- #define SC_R_VPU_UART 366U
- #define SC_R_VPUCORE 367U
- #define SC_R_VPUCORE_0 368U
- #define SC_R_VPUCORE_1 369U
- #define SC_R_VPUCORE_2 370U
- #define SC_R_VPUCORE_3 371U
- #define SC_R_DMA_4_CH0 372U
- #define SC_R_DMA_4_CH1 373U
- #define SC_R_DMA_4_CH2 374U
- #define SC_R_DMA_4_CH3 375U
- #define SC_R_DMA_4_CH4 376U
- #define SC_R_ISI_CH0 377U
- #define SC_R_ISI_CH1 378U
- #define SC_R_ISI_CH2 379U
- #define SC_R_ISI_CH3 380U
- #define SC_R_ISI_CH4 381U
- #define SC_R_ISI_CH5 382U
- #define SC_R_ISI_CH6 383U
- #define SC_R_ISI_CH7 384U
- #define SC_R_MJPEG_DEC_S0 385U
- #define SC_R_MJPEG_DEC_S1 386U
- #define SC_R_MJPEG_DEC_S2 387U
- #define SC_R_MJPEG_DEC_S3 388U
- #define SC_R_MJPEG_ENC_S0 389U
- #define SC_R_MJPEG_ENC_S1 390U
- #define SC_R_MJPEG_ENC_S2 391U
- #define SC_R_MJPEG_ENC_S3 392U
- #define SC_R_MIPI_0 393U

- #define SC_R_MIPI_0_PWM_0 394U
- #define SC_R_MIPI_0_I2C_0 395U
- #define SC_R_MIPI_0_I2C_1 396U
- #define SC_R_MIPI_1 397U
- #define SC_R_MIPI_1_PWM_0 398U
- #define SC_R_MIPI_1_I2C_0 399U
- #define SC_R_MIPI_1_I2C_1 400U
- #define SC_R_CSI_0 401U
- #define SC_R_CSI_0_PWM_0 402U
- #define SC_R_CSI_0_I2C_0 403U
- #define SC_R_CSI_1 404U
- #define SC_R_CSI_1_PWM_0 405U
- #define SC_R_CSI_1_I2C_0 406U
- #define SC_R_HDMI 407U
- #define SC_R_HDMI_I2S 408U
- #define SC_R_HDMI_I2C_0 409U
- #define SC_R_HDMI_PLL_0 410U
- #define SC_R_HDMI_RX 411U
- #define SC_R_HDMI_RX_BYPASS 412U
- #define SC_R_HDMI_RX_I2C_0 413U
- #define SC_R_ASRC_0 414U
- #define SC_R_ESAI_0 415U
- #define SC_R_SPDIF_0 416U
- #define SC_R_SPDIF_1 417U
- #define SC_R_SAI_3 418U
- #define SC_R_SAI_4 419U
- #define SC_R_SAI_5 420U
- #define SC_R_GPT_5 421U
- #define SC_R_GPT_6 422U
- #define SC_R_GPT_7 423U
- #define SC_R_GPT_8 424U
- #define SC_R_GPT_9 425U
- #define SC_R_GPT_10 426U
- #define SC_R_DMA_2_CH5 427U
- #define SC_R_DMA_2_CH6 428U
- #define SC_R_DMA_2_CH7 429U
- #define SC_R_DMA_2_CH8 430U
- #define SC_R_DMA_2_CH9 431U
- #define SC_R_DMA_2_CH10 432U
- #define SC_R_DMA_2_CH11 433U
- #define SC_R_DMA_2_CH12 434U
- #define SC_R_DMA_2_CH13 435U
- #define SC_R_DMA_2_CH14 436U
- #define SC_R_DMA_2_CH15 437U
- #define SC_R_DMA_2_CH16 438U
- #define SC_R_DMA_2_CH17 439U
- #define SC_R_DMA_2_CH18 440U
- #define SC_R_DMA_2_CH19 441U
- #define SC_R_DMA_2_CH20 442U
- #define SC_R_DMA_2_CH21 443U
- #define SC_R_DMA_2_CH22 444U
- #define SC_R_DMA_2_CH23 445U
- #define SC_R_DMA_2_CH24 446U
- #define SC_R_DMA_2_CH25 447U
- #define SC_R_DMA_2_CH26 448U
- #define SC_R_DMA_2_CH27 449U
- #define SC_R_DMA_2_CH28 450U
- #define SC_R_DMA_2_CH29 451U
- #define SC_R_DMA_2_CH30 452U
- #define SC_R_DMA_2_CH31 453U

- #define **SC_R_ASRC_1** 454U
- #define **SC_R_ESAI_1** 455U
- #define **SC_R_SAI_6** 456U
- #define **SC_R_SAI_7** 457U
- #define **SC_R_AMIX** 458U
- #define **SC_R_MQS_0** 459U
- #define **SC_R_DMA_3_CH0** 460U
- #define **SC_R_DMA_3_CH1** 461U
- #define **SC_R_DMA_3_CH2** 462U
- #define **SC_R_DMA_3_CH3** 463U
- #define **SC_R_DMA_3_CH4** 464U
- #define **SC_R_DMA_3_CH5** 465U
- #define **SC_R_DMA_3_CH6** 466U
- #define **SC_R_DMA_3_CH7** 467U
- #define **SC_R_DMA_3_CH8** 468U
- #define **SC_R_DMA_3_CH9** 469U
- #define **SC_R_DMA_3_CH10** 470U
- #define **SC_R_DMA_3_CH11** 471U
- #define **SC_R_DMA_3_CH12** 472U
- #define **SC_R_DMA_3_CH13** 473U
- #define **SC_R_DMA_3_CH14** 474U
- #define **SC_R_DMA_3_CH15** 475U
- #define **SC_R_DMA_3_CH16** 476U
- #define **SC_R_DMA_3_CH17** 477U
- #define **SC_R_DMA_3_CH18** 478U
- #define **SC_R_DMA_3_CH19** 479U
- #define **SC_R_DMA_3_CH20** 480U
- #define **SC_R_DMA_3_CH21** 481U
- #define **SC_R_DMA_3_CH22** 482U
- #define **SC_R_DMA_3_CH23** 483U
- #define **SC_R_DMA_3_CH24** 484U
- #define **SC_R_DMA_3_CH25** 485U
- #define **SC_R_DMA_3_CH26** 486U
- #define **SC_R_DMA_3_CH27** 487U
- #define **SC_R_DMA_3_CH28** 488U
- #define **SC_R_DMA_3_CH29** 489U
- #define **SC_R_DMA_3_CH30** 490U
- #define **SC_R_DMA_3_CH31** 491U
- #define **SC_R_AUDIO_PLL_1** 492U
- #define **SC_R_AUDIO_CLK_0** 493U
- #define **SC_R_AUDIO_CLK_1** 494U
- #define **SC_R_MCLK_OUT_0** 495U
- #define **SC_R_MCLK_OUT_1** 496U
- #define **SC_R_PMIC_0** 497U
- #define **SC_R_PMIC_1** 498U
- #define **SC_R_SECO** 499U
- #define **SC_R_CAAM_JR1** 500U
- #define **SC_R_CAAM_JR2** 501U
- #define **SC_R_CAAM_JR3** 502U
- #define **SC_R_SECO_MU_2** 503U
- #define **SC_R_SECO_MU_3** 504U
- #define **SC_R_SECO_MU_4** 505U
- #define **SC_R_HDMI_RX_PWM_0** 506U
- #define **SC_R_A35** 507U
- #define **SC_R_A35_0** 508U
- #define **SC_R_A35_1** 509U
- #define **SC_R_A35_2** 510U
- #define **SC_R_A35_3** 511U
- #define **SC_R_DSP** 512U
- #define **SC_R_DSP_RAM** 513U

- #define **SC_R_CAAM_JR1_OUT** 514U
- #define **SC_R_CAAM_JR2_OUT** 515U
- #define **SC_R_CAAM_JR3_OUT** 516U
- #define **SC_R_VPU_DEC_0** 517U
- #define **SC_R_VPU_ENC_0** 518U
- #define **SC_R_CAAM_JR0** 519U
- #define **SC_R_CAAM_JR0_OUT** 520U
- #define **SC_R_PMIC_2** 521U
- #define **SC_R_DBLOGIC** 522U
- #define **SC_R_HDMI_PLL_1** 523U
- #define **SC_R_BOARD_R0** 524U
- #define **SC_R_BOARD_R1** 525U
- #define **SC_R_BOARD_R2** 526U
- #define **SC_R_BOARD_R3** 527U
- #define **SC_R_BOARD_R4** 528U
- #define **SC_R_BOARD_R5** 529U
- #define **SC_R_BOARD_R6** 530U
- #define **SC_R_BOARD_R7** 531U
- #define **SC_R_MJPEG_DEC_MP** 532U
- #define **SC_R_MJPEG_ENC_MP** 533U
- #define **SC_R_VPU_TS_0** 534U
- #define **SC_R_VPU_MU_0** 535U
- #define **SC_R_VPU_MU_1** 536U
- #define **SC_R_VPU_MU_2** 537U
- #define **SC_R_VPU_MU_3** 538U
- #define **SC_R_VPU_ENC_1** 539U
- #define **SC_R_VPU** 540U
- #define **SC_R_DMA_5_CH0** 541U
- #define **SC_R_DMA_5_CH1** 542U
- #define **SC_R_DMA_5_CH2** 543U
- #define **SC_R_DMA_5_CH3** 544U
- #define **SC_R_ATTESTATION** 545U
- #define **SC_R_LAST** 546U
- #define **SC_R_ALL** ((**sc_rsrc_t**) UINT16_MAX)

All resources.

Typedefs

- typedef **uint8_t sc_bool_t**
This type is used to store a boolean.
- typedef **uint64_t sc_faddr_t**
This type is used to store a system (full-size) address.
- typedef **uint8_t sc_err_t**
This type is used to indicate error response for most functions.
- typedef **uint16_t sc_rsrc_t**
This type is used to indicate a resource.
- typedef **uint8_t sc_ctrl_t**
This type is used to indicate a control.
- typedef **uint16_t sc_pad_t**
This type is used to indicate a pad.
- typedef **__INT8_TYPE__ int8_t**
Type used to declare an 8-bit integer.
- typedef **__INT16_TYPE__ int16_t**
Type used to declare a 16-bit integer.

- `typedef __INT32_TYPE__ int32_t`
Type used to declare a 32-bit integer.
- `typedef __INT64_TYPE__ int64_t`
Type used to declare a 64-bit integer.
- `typedef __UINT8_TYPE__ uint8_t`
Type used to declare an 8-bit unsigned integer.
- `typedef __UINT16_TYPE__ uint16_t`
Type used to declare a 16-bit unsigned integer.
- `typedef __UINT32_TYPE__ uint32_t`
Type used to declare a 32-bit unsigned integer.
- `typedef __UINT64_TYPE__ uint64_t`
Type used to declare a 64-bit unsigned integer.

11.2.1 Detailed Description

Header file containing types used across multiple service APIs.

11.2.2 Typedef Documentation

11.2.2.1 `sc_rsrc_t`

```
typedef uint16_t sc_rsrc_t
```

This type is used to indicate a resource.

Resources include peripherals and bus masters (but not memory regions). Note items from list should never be changed or removed (only added to at the end of the list).

11.2.2.2 `sc_pad_t`

```
typedef uint16_t sc_pad_t
```

This type is used to indicate a pad.

Valid values are SoC specific.

Refer to the SoC [Pad List](#) for valid pad values.

11.3 `platform/svc/irq/api.h` File Reference

Header file containing the public API for the System Controller (SC) Interrupt (IRQ) function.

Macros

- #define `SC_IRQ_NUM_GROUP` 5U
Number of groups.

Defines for `sc_irq_group_t`

- #define `SC_IRQ_GROUP_TEMP` 0U
Temp interrupts.
- #define `SC_IRQ_GROUP_WDOG` 1U
Watchdog interrupts.
- #define `SC_IRQ_GROUP_RTC` 2U
RTC interrupts.
- #define `SC_IRQ_GROUP_WAKE` 3U
Wakeup interrupts.
- #define `SC_IRQ_GROUP_SYSCTR` 4U
System counter interrupts.

Defines for `sc_irq_temp_t`

- #define `SC_IRQ_TEMP_HIGH` (1UL << 0U)
Temp alarm interrupt.
- #define `SC_IRQ_TEMP_CPU0_HIGH` (1UL << 1U)
CPU0 temp alarm interrupt.
- #define `SC_IRQ_TEMP_CPU1_HIGH` (1UL << 2U)
CPU1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_GPU0_HIGH` (1UL << 3U)
GPU0 temp alarm interrupt.
- #define `SC_IRQ_TEMP_GPU1_HIGH` (1UL << 4U)
GPU1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_DRC0_HIGH` (1UL << 5U)
DRC0 temp alarm interrupt.
- #define `SC_IRQ_TEMP_DRC1_HIGH` (1UL << 6U)
DRC1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_VPU_HIGH` (1UL << 7U)
DRC1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_PMIC0_HIGH` (1UL << 8U)
PMIC0 temp alarm interrupt.
- #define `SC_IRQ_TEMP_PMIC1_HIGH` (1UL << 9U)
PMIC1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_LOW` (1UL << 10U)
Temp alarm interrupt.
- #define `SC_IRQ_TEMP_CPU0_LOW` (1UL << 11U)
CPU0 temp alarm interrupt.
- #define `SC_IRQ_TEMP_CPU1_LOW` (1UL << 12U)
CPU1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_GPU0_LOW` (1UL << 13U)
GPU0 temp alarm interrupt.
- #define `SC_IRQ_TEMP_GPU1_LOW` (1UL << 14U)
GPU1 temp alarm interrupt.
- #define `SC_IRQ_TEMP_DRC0_LOW` (1UL << 15U)
DRC0 temp alarm interrupt.
- #define `SC_IRQ_TEMP_DRC1_LOW` (1UL << 16U)

- *DRC1 temp alarm interrupt.*
• #define `SC_IRQ_TEMP_VPU_LOW` (1UL << 17U)
- *DRC1 temp alarm interrupt.*
• #define `SC_IRQ_TEMP_PMIC0_LOW` (1UL << 18U)
- *PMIC0 temp alarm interrupt.*
• #define `SC_IRQ_TEMP_PMIC1_LOW` (1UL << 19U)
- *PMIC1 temp alarm interrupt.*
• #define `SC_IRQ_TEMP_PMIC2_HIGH` (1UL << 20U)
- *PMIC2 temp alarm interrupt.*
• #define `SC_IRQ_TEMP_PMIC2_LOW` (1UL << 21U)
- *PMIC2 temp alarm interrupt.*

Defines for `sc_irq_wdog_t`

- #define `SC_IRQ_WDOG` (1U << 0U)
Watchdog interrupt.

Defines for `sc_irq_rtc_t`

- #define `SC_IRQ_RTC` (1U << 0U)
RTC interrupt.

Defines for `sc_irq_wake_t`

- #define `SC_IRQ_BUTTON` (1U << 0U)
Button interrupt.
- #define `SC_IRQ_PAD` (1U << 1U)
Pad wakeup.

Defines for `sc_irq_sysctr_t`

- #define `SC_IRQ_SYSCTR` (1U << 0U)
SYSCTR interrupt.

Typedefs

- typedef `uint8_t sc_irq_group_t`
This type is used to declare an interrupt group.
- typedef `uint8_t sc_irq_temp_t`
This type is used to declare a bit mask of temp interrupts.
- typedef `uint8_t sc_irq_wdog_t`
This type is used to declare a bit mask of watchdog interrupts.
- typedef `uint8_t sc_irq_rtc_t`
This type is used to declare a bit mask of RTC interrupts.
- typedef `uint8_t sc_irq_wake_t`
This type is used to declare a bit mask of wakeup interrupts.

Functions

- `sc_err_t sc_irq_enable` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_irq_group_t` group, `uint32_t` mask, `sc_bool_t` enable)
This function enables/disables interrupts.
- `sc_err_t sc_irq_status` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_irq_group_t` group, `uint32_t` *status)
This function returns the current interrupt status (regardless if masked).

11.3.1 Detailed Description

Header file containing the public API for the System Controller (SC) Interrupt (IRQ) function.

11.4 platform/svc/misc/api.h File Reference

Header file containing the public API for the System Controller (SC) Miscellaneous (MISC) function.

Macros

- `#define SC_MISC_DMA_GRP_MAX` 31U
Max DMA channel priority group.

Defines for type widths

- `#define SC_MISC_DMA_GRP_W` 5U
Width of `sc_misc_dma_group_t`.

Defines for `sc_misc_boot_status_t`

- `#define SC_MISC_BOOT_STATUS_SUCCESS` 0U
Success.
- `#define SC_MISC_BOOT_STATUS_SECURITY` 1U
Security violation.

Defines for `sc_misc_temp_t`

- `#define SC_MISC_TEMP` 0U
Temp sensor.
- `#define SC_MISC_TEMP_HIGH` 1U
Temp high alarm.
- `#define SC_MISC_TEMP_LOW` 2U
Temp low alarm.

Defines for `sc_misc_seco_auth_cmd_t`

- `#define SC_MISC_AUTH_CONTAINER` 0U
Authenticate container.

- `#define SC_MISC_VERIFY_IMAGE 1U`
Verify image.
- `#define SC_MISC_REL_CONTAINER 2U`
Release container.
- `#define SC_MISC_SECO_AUTH_SECO_FW 3U`
SECO Firmware.
- `#define SC_MISC_SECO_AUTH_HDMI_TX_FW 4U`
HDMI TX Firmware.
- `#define SC_MISC_SECO_AUTH_HDMI_RX_FW 5U`
HDMI RX Firmware.

Defines for `sc_misc_bt_t`

- `#define SC_MISC_BT_PRIMARY 0U`
- `#define SC_MISC_BT_SECONDARY 1U`
- `#define SC_MISC_BT_RECOVERY 2U`
- `#define SC_MISC_BT_MANUFACTURE 3U`
- `#define SC_MISC_BT_SERIAL 4U`

Typedefs

- `typedef uint8_t sc_misc_dma_group_t`
This type is used to store a DMA channel priority group.
- `typedef uint8_t sc_misc_boot_status_t`
This type is used report boot status.
- `typedef uint8_t sc_misc_seco_auth_cmd_t`
This type is used to issue SECO authenticate commands.
- `typedef uint8_t sc_misc_temp_t`
This type is used report boot status.
- `typedef uint8_t sc_misc_bt_t`
This type is used report the boot type.

Functions

Control Functions

- `sc_err_t sc_misc_set_control` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_ctrl_t` ctrl, `uint32_t` val)
This function sets a miscellaneous control value.
- `sc_err_t sc_misc_get_control` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_ctrl_t` ctrl, `uint32_t` *val)
This function gets a miscellaneous control value.

DMA Functions

- `sc_err_t sc_misc_set_max_dma_group` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt, `sc_misc_dma_group_t` max)
This function configures the max DMA channel priority group for a partition.
- `sc_err_t sc_misc_set_dma_group` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_misc_dma_group_t` group)
This function configures the priority group for a DMA channel.

Security Functions

- `sc_err_t sc_misc_seco_image_load` (`sc_ipc_t ipc`, `sc_faddr_t addr_src`, `sc_faddr_t addr_dst`, `uint32_t len`, `sc_bool_t fw`)
This function loads a SECO image.
- `sc_err_t sc_misc_seco_authenticate` (`sc_ipc_t ipc`, `sc_misc_seco_auth_cmd_t cmd`, `sc_faddr_t addr`)
This function is used to authenticate a SECO image or command.
- `sc_err_t sc_misc_seco_fuse_write` (`sc_ipc_t ipc`, `sc_faddr_t addr`)
This function securely writes a group of fuse words.
- `sc_err_t sc_misc_seco_enable_debug` (`sc_ipc_t ipc`, `sc_faddr_t addr`)
This function securely enables debug.
- `sc_err_t sc_misc_seco_forward_lifecycle` (`sc_ipc_t ipc`, `uint32_t change`)
This function updates the lifecycle of the device.
- `sc_err_t sc_misc_seco_return_lifecycle` (`sc_ipc_t ipc`, `sc_faddr_t addr`)
This function updates the lifecycle to one of the return lifecycles.
- `void sc_misc_seco_build_info` (`sc_ipc_t ipc`, `uint32_t *version`, `uint32_t *commit`)
This function is used to return the SECO FW build info.
- `sc_err_t sc_misc_seco_chip_info` (`sc_ipc_t ipc`, `uint16_t *lc`, `uint16_t *monotonic`, `uint32_t *uid_l`, `uint32_t *uid_h`)
This function is used to return SECO chip info.
- `sc_err_t sc_misc_seco_attest_mode` (`sc_ipc_t ipc`, `uint32_t mode`)
This function is used to set the attestation mode.
- `sc_err_t sc_misc_seco_attest` (`sc_ipc_t ipc`, `uint64_t nonce`)
This function is used to request atestation.
- `sc_err_t sc_misc_seco_get_attest_pkey` (`sc_ipc_t ipc`, `sc_faddr_t addr`)
This function is used to retrieve the attestation public key.
- `sc_err_t sc_misc_seco_get_attest_sign` (`sc_ipc_t ipc`, `sc_faddr_t addr`)
This function is used to retrieve attestation signature and parameters.
- `sc_err_t sc_misc_seco_attest_verify` (`sc_ipc_t ipc`, `sc_faddr_t addr`)
This function is used to verify attestation.
- `sc_err_t sc_misc_seco_commit` (`sc_ipc_t ipc`, `uint32_t *info`)
This function is used to commit into the fuses any new SRK revocation and FW version information that have been found in the primary and secondary containers.

Debug Functions

- `void sc_misc_debug_out` (`sc_ipc_t ipc`, `uint8_t ch`)
This function is used output a debug character from the SCU UART.
- `sc_err_t sc_misc_waveform_capture` (`sc_ipc_t ipc`, `sc_bool_t enable`)
This function starts/stops emulation waveform capture.
- `void sc_misc_build_info` (`sc_ipc_t ipc`, `uint32_t *build`, `uint32_t *commit`)
This function is used to return the SCFW build info.
- `void sc_misc_unique_id` (`sc_ipc_t ipc`, `uint32_t *id_l`, `uint32_t *id_h`)
This function is used to return the device's unique ID.

Other Functions

- `sc_err_t sc_misc_set_ari` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_rsrc_t resource_mst`, `uint16_t ari`, `sc_bool_t enable`)
This function configures the ARI match value for PCIe/SATA resources.
- `void sc_misc_boot_status` (`sc_ipc_t ipc`, `sc_misc_boot_status_t status`)
This function reports boot status.
- `sc_err_t sc_misc_boot_done` (`sc_ipc_t ipc`, `sc_rsrc_t cpu`)
This function tells the SCFW that a CPU is done booting.
- `sc_err_t sc_misc_otp_fuse_read` (`sc_ipc_t ipc`, `uint32_t word`, `uint32_t *val`)

- This function reads a given fuse word index.*
- `sc_err_t sc_misc_otp_fuse_write` (`sc_ipc_t ipc`, `uint32_t word`, `uint32_t val`)
- This function writes a given fuse word index.*
- `sc_err_t sc_misc_set_temp` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_misc_temp_t temp`, `int16_t celsius`, `int8_t tenths`)
- This function sets a temp sensor alarm.*
- `sc_err_t sc_misc_get_temp` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_misc_temp_t temp`, `int16_t *celsius`, `int8_t *tenths`)
- This function gets a temp sensor value.*
- `void sc_misc_get_boot_dev` (`sc_ipc_t ipc`, `sc_rsrc_t *dev`)
- This function returns the boot device.*
- `sc_err_t sc_misc_get_boot_type` (`sc_ipc_t ipc`, `sc_misc_bt_t *type`)
- This function returns the boot type.*
- `void sc_misc_get_button_status` (`sc_ipc_t ipc`, `sc_bool_t *status`)
- This function returns the current status of the ON/OFF button.*
- `sc_err_t sc_misc_rompatch_checksum` (`sc_ipc_t ipc`, `uint32_t *checksum`)
- This function returns the ROM patch checksum.*

11.4.1 Detailed Description

Header file containing the public API for the System Controller (SC) Miscellaneous (MISC) function.

11.5 platform/svc/pad/api.h File Reference

Header file containing the public API for the System Controller (SC) Pad Control (PAD) function.

Macros

Defines for type widths

- `#define SC_PAD_MUX_W` 3U
- Width of mux parameter.*

Defines for `sc_pad_config_t`

- `#define SC_PAD_CONFIG_NORMAL` 0U
- Normal.*
- `#define SC_PAD_CONFIG_OD` 1U
- Open Drain.*
- `#define SC_PAD_CONFIG_OD_IN` 2U
- Open Drain and input.*
- `#define SC_PAD_CONFIG_OUT_IN` 3U
- Output and input.*

Defines for `sc_pad_iso_t`

- `#define SC_PAD_ISO_OFF` 0U

- *ISO latch is transparent.*
- #define `SC_PAD_ISO_EARLY` 1U
Follow EARLY_ISO.
- #define `SC_PAD_ISO_LATE` 2U
Follow LATE_ISO.
- #define `SC_PAD_ISO_ON` 3U
ISO latched data is held.

Defines for `sc_pad_28fdsoi_dse_t`

- #define `SC_PAD_28FDSOI_DSE_18V_1MA` 0U
Drive strength of 1mA for 1.8v.
- #define `SC_PAD_28FDSOI_DSE_18V_2MA` 1U
Drive strength of 2mA for 1.8v.
- #define `SC_PAD_28FDSOI_DSE_18V_4MA` 2U
Drive strength of 4mA for 1.8v.
- #define `SC_PAD_28FDSOI_DSE_18V_6MA` 3U
Drive strength of 6mA for 1.8v.
- #define `SC_PAD_28FDSOI_DSE_18V_8MA` 4U
Drive strength of 8mA for 1.8v.
- #define `SC_PAD_28FDSOI_DSE_18V_10MA` 5U
Drive strength of 10mA for 1.8v.
- #define `SC_PAD_28FDSOI_DSE_18V_12MA` 6U
Drive strength of 12mA for 1.8v.
- #define `SC_PAD_28FDSOI_DSE_18V_HS` 7U
High-speed drive strength for 1.8v.
- #define `SC_PAD_28FDSOI_DSE_33V_2MA` 0U
Drive strength of 2mA for 3.3v.
- #define `SC_PAD_28FDSOI_DSE_33V_4MA` 1U
Drive strength of 4mA for 3.3v.
- #define `SC_PAD_28FDSOI_DSE_33V_8MA` 2U
Drive strength of 8mA for 3.3v.
- #define `SC_PAD_28FDSOI_DSE_33V_12MA` 3U
Drive strength of 12mA for 3.3v.
- #define `SC_PAD_28FDSOI_DSE_DV_HIGH` 0U
High drive strength for dual volt.
- #define `SC_PAD_28FDSOI_DSE_DV_LOW` 1U
Low drive strength for dual volt.

Defines for `sc_pad_28fdsoi_ps_t`

- #define `SC_PAD_28FDSOI_PS_KEEPER` 0U
Bus-keeper (only valid for 1.8v)
- #define `SC_PAD_28FDSOI_PS_PU` 1U
Pull-up.
- #define `SC_PAD_28FDSOI_PS_PD` 2U
Pull-down.
- #define `SC_PAD_28FDSOI_PS_NONE` 3U
No pull (disabled)

Defines for `sc_pad_28fdsoi_pus_t`

- #define `SC_PAD_28FDSOI_PUS_30K_PD` 0U

- *30K pull-down*
• #define SC_PAD_28FDSOI_PUS_100K_PU 1U
- *100K pull-up*
• #define SC_PAD_28FDSOI_PUS_3K_PU 2U
- *3K pull-up*
• #define SC_PAD_28FDSOI_PUS_30K_PU 3U
- *30K pull-up*

Defines for sc_pad_wakeup_t

- #define SC_PAD_WAKEUP_OFF 0U
Off.
- #define SC_PAD_WAKEUP_CLEAR 1U
Clears pending flag.
- #define SC_PAD_WAKEUP_LOW_LVL 4U
Low level.
- #define SC_PAD_WAKEUP_FALL_EDGE 5U
Falling edge.
- #define SC_PAD_WAKEUP_RISE_EDGE 6U
Rising edge.
- #define SC_PAD_WAKEUP_HIGH_LVL 7U
High-level.

Typedefs

- typedef uint8_t sc_pad_config_t
This type is used to declare a pad config.
- typedef uint8_t sc_pad_iso_t
This type is used to declare a pad low-power isolation config.
- typedef uint8_t sc_pad_28fdsoi_dse_t
This type is used to declare a drive strength.
- typedef uint8_t sc_pad_28fdsoi_ps_t
This type is used to declare a pull select.
- typedef uint8_t sc_pad_28fdsoi_pus_t
This type is used to declare a pull-up select.
- typedef uint8_t sc_pad_wakeup_t
This type is used to declare a wakeup mode of a pad.

Functions

Generic Functions

- sc_err_t sc_pad_set_mux (sc_ipc_t ipc, sc_pad_t pad, uint8_t mux, sc_pad_config_t config, sc_pad_iso_t iso)
This function configures the mux settings for a pad.
- sc_err_t sc_pad_get_mux (sc_ipc_t ipc, sc_pad_t pad, uint8_t *mux, sc_pad_config_t *config, sc_pad_iso_t *iso)
This function gets the mux settings for a pad.
- sc_err_t sc_pad_set_gp (sc_ipc_t ipc, sc_pad_t pad, uint32_t ctrl)
This function configures the general purpose pad control.
- sc_err_t sc_pad_get_gp (sc_ipc_t ipc, sc_pad_t pad, uint32_t *ctrl)

This function gets the general purpose pad control.

- `sc_err_t sc_pad_set_wakeup` (`sc_ipc_t` ipc, `sc_pad_t` pad, `sc_pad_wakeup_t` wakeup)

This function configures the wakeup mode of the pad.

- `sc_err_t sc_pad_get_wakeup` (`sc_ipc_t` ipc, `sc_pad_t` pad, `sc_pad_wakeup_t` *wakeup)

This function gets the wakeup mode of a pad.

- `sc_err_t sc_pad_set_all` (`sc_ipc_t` ipc, `sc_pad_t` pad, `uint8_t` mux, `sc_pad_config_t` config, `sc_pad_iso_t` iso, `uint32_t` ctrl, `sc_pad_wakeup_t` wakeup)

This function configures a pad.

- `sc_err_t sc_pad_get_all` (`sc_ipc_t` ipc, `sc_pad_t` pad, `uint8_t` *mux, `sc_pad_config_t` *config, `sc_pad_iso_t` *iso, `uint32_t` *ctrl, `sc_pad_wakeup_t` *wakeup)

This function gets a pad's config.

SoC Specific Functions

- `sc_err_t sc_pad_set` (`sc_ipc_t` ipc, `sc_pad_t` pad, `uint32_t` val)

This function configures the settings for a pad.

- `sc_err_t sc_pad_get` (`sc_ipc_t` ipc, `sc_pad_t` pad, `uint32_t` *val)

This function gets the settings for a pad.

Technology Specific Functions

- `sc_err_t sc_pad_set_gp_28fdsoi` (`sc_ipc_t` ipc, `sc_pad_t` pad, `sc_pad_28fdsoi_dse_t` dse, `sc_pad_28fdsoi_ps_t` ps)

This function configures the pad control specific to 28FDSOI.

- `sc_err_t sc_pad_get_gp_28fdsoi` (`sc_ipc_t` ipc, `sc_pad_t` pad, `sc_pad_28fdsoi_dse_t` *dse, `sc_pad_28fdsoi_ps_t` *ps)

This function gets the pad control specific to 28FDSOI.

- `sc_err_t sc_pad_set_gp_28fdsoi_hsic` (`sc_ipc_t` ipc, `sc_pad_t` pad, `sc_pad_28fdsoi_dse_t` dse, `sc_bool_t` hys, `sc_pad_28fdsoi_pus_t` pus, `sc_bool_t` pke, `sc_bool_t` pue)

This function configures the pad control specific to 28FDSOI.

- `sc_err_t sc_pad_get_gp_28fdsoi_hsic` (`sc_ipc_t` ipc, `sc_pad_t` pad, `sc_pad_28fdsoi_dse_t` *dse, `sc_bool_t` *hys, `sc_pad_28fdsoi_pus_t` *pus, `sc_bool_t` *pke, `sc_bool_t` *pue)

This function gets the pad control specific to 28FDSOI.

- `sc_err_t sc_pad_set_gp_28fdsoi_comp` (`sc_ipc_t` ipc, `sc_pad_t` pad, `uint8_t` compen, `sc_bool_t` fastfrz, `uint8_t` rasrcp, `uint8_t` rasrcn, `sc_bool_t` nasrc_sel, `sc_bool_t` psw_ovr)

This function configures the compensation control specific to 28FDSOI.

- `sc_err_t sc_pad_get_gp_28fdsoi_comp` (`sc_ipc_t` ipc, `sc_pad_t` pad, `uint8_t` *compen, `sc_bool_t` *fastfrz, `uint8_t` *rasrcp, `uint8_t` *rasrcn, `sc_bool_t` *nasrc_sel, `sc_bool_t` *compok, `uint8_t` *nasrc, `sc_bool_t` *psw_ovr)

This function gets the compensation control specific to 28FDSOI.

11.5.1 Detailed Description

Header file containing the public API for the System Controller (SC) Pad Control (PAD) function.

11.6 platform/svc/pm/api.h File Reference

Header file containing the public API for the System Controller (SC) Power Management (PM) function.

Macros

Defines for type widths

- #define `SC_PM_POWER_MODE_W` 2U
Width of `sc_pm_power_mode_t`.
- #define `SC_PM_CLOCK_MODE_W` 3U
Width of `sc_pm_clock_mode_t`.
- #define `SC_PM_RESET_TYPE_W` 2U
Width of `sc_pm_reset_type_t`.
- #define `SC_PM_RESET_REASON_W` 4U
Width of `sc_pm_reset_reason_t`.

Defines for ALL parameters

- #define `SC_PM_CLK_ALL` ((`sc_pm_clk_t`) UINT8_MAX)
All clocks.

Defines for `sc_pm_power_mode_t`

- #define `SC_PM_PW_MODE_OFF` 0U
Power off.
- #define `SC_PM_PW_MODE_STBY` 1U
Power in standby.
- #define `SC_PM_PW_MODE_LP` 2U
Power in low-power.
- #define `SC_PM_PW_MODE_ON` 3U
Power on.

Defines for `sc_pm_clk_t`

- #define `SC_PM_CLK_SLV_BUS` 0U
Slave bus clock.
- #define `SC_PM_CLK_MST_BUS` 1U
Master bus clock.
- #define `SC_PM_CLK_PER` 2U
Peripheral clock.
- #define `SC_PM_CLK_PHY` 3U
Phy clock.
- #define `SC_PM_CLK_MISC` 4U
Misc clock.
- #define `SC_PM_CLK_MISC0` 0U
Misc 0 clock.
- #define `SC_PM_CLK_MISC1` 1U
Misc 1 clock.
- #define `SC_PM_CLK_MISC2` 2U
Misc 2 clock.
- #define `SC_PM_CLK_MISC3` 3U
Misc 3 clock.
- #define `SC_PM_CLK_MISC4` 4U
Misc 4 clock.
- #define `SC_PM_CLK_CPU` 2U
CPU clock.

- #define `SC_PM_CLK_PLL` 4U
PLL.
- #define `SC_PM_CLK_BYPASS` 4U
Bypass clock.

Defines for `sc_pm_clk_mode_t`

- #define `SC_PM_CLK_MODE_ROM_INIT` 0U
Clock is initialized by ROM.
- #define `SC_PM_CLK_MODE_OFF` 1U
Clock is disabled.
- #define `SC_PM_CLK_MODE_ON` 2U
Clock is enabled.
- #define `SC_PM_CLK_MODE_AUTOGATE_SW` 3U
Clock is in SW autogate mode.
- #define `SC_PM_CLK_MODE_AUTOGATE_HW` 4U
Clock is in HW autogate mode.
- #define `SC_PM_CLK_MODE_AUTOGATE_SW_HW` 5U
Clock is in SW-HW autogate mode.

Defines for `sc_pm_clk_parent_t`

- #define `SC_PM_PARENT_XTAL` 0U
Parent is XTAL.
- #define `SC_PM_PARENT_PLL0` 1U
Parent is PLL0.
- #define `SC_PM_PARENT_PLL1` 2U
Parent is PLL1 or PLL0/2.
- #define `SC_PM_PARENT_PLL2` 3U
Parent in PLL2 or PLL0/4.
- #define `SC_PM_PARENT_BYPS` 4U
Parent is a bypass clock.

Defines for `sc_pm_reset_type_t`

- #define `SC_PM_RESET_TYPE_COLD` 0U
Cold reset.
- #define `SC_PM_RESET_TYPE_WARM` 1U
Warm reset.
- #define `SC_PM_RESET_TYPE_BOARD` 2U
Board reset.

Defines for `sc_pm_reset_reason_t`

- #define `SC_PM_RESET_REASON_POR` 0U
Power on reset.
- #define `SC_PM_RESET_REASON_JTAG` 1U
JTAG reset.
- #define `SC_PM_RESET_REASON_SW` 2U
Software reset.
- #define `SC_PM_RESET_REASON_WDOG` 3U
Partition watchdog reset.

- `#define SC_PM_RESET_REASON_LOCKUP 4U`
SCU lockup reset.
- `#define SC_PM_RESET_REASON_SNVS 5U`
SNVS reset.
- `#define SC_PM_RESET_REASON_TEMP 6U`
Temp panic reset.
- `#define SC_PM_RESET_REASON_MSI 7U`
MSI reset.
- `#define SC_PM_RESET_REASON_UECC 8U`
ECC reset.
- `#define SC_PM_RESET_REASON_SCFW_WDOG 9U`
SCFW watchdog reset.
- `#define SC_PM_RESET_REASON_ROM_WDOG 10U`
SCU ROM watchdog reset.
- `#define SC_PM_RESET_REASON_SECO 11U`
SECO reset.
- `#define SC_PM_RESET_REASON_SCFW_FAULT 12U`
SCFW fault reset.

Defines for `sc_pm_sys_if_t`

- `#define SC_PM_SYS_IF_INTERCONNECT 0U`
System interconnect.
- `#define SC_PM_SYS_IF_MU 1U`
AP -> SCU message units.
- `#define SC_PM_SYS_IF_OCMEM 2U`
On-chip memory (ROM/OCRAM)
- `#define SC_PM_SYS_IF_DDR 3U`
DDR memory.

Defines for `sc_pm_wake_src_t`

- `#define SC_PM_WAKE_SRC_NONE 0U`
No wake source, used for self-kill.
- `#define SC_PM_WAKE_SRC_SCU 1U`
Wakeup from SCU to resume CPU (IRQSTEER & GIC powered down)
- `#define SC_PM_WAKE_SRC_IRQSTEER 2U`
Wakeup from IRQSTEER to resume CPU (GIC powered down)
- `#define SC_PM_WAKE_SRC_IRQSTEER_GIC 3U`
Wakeup from IRQSTEER+GIC to wake CPU (GIC clock gated)
- `#define SC_PM_WAKE_SRC_GIC 4U`
Wakeup from GIC to wake CPU.

Typedefs

- `typedef uint8_t sc_pm_power_mode_t`
This type is used to declare a power mode.
- `typedef uint8_t sc_pm_clk_t`
This type is used to declare a clock.
- `typedef uint8_t sc_pm_clk_mode_t`
This type is used to declare a clock mode.

- typedef [uint8_t sc_pm_clk_parent_t](#)
This type is used to declare the clock parent.
- typedef [uint32_t sc_pm_clock_rate_t](#)
This type is used to declare clock rates.
- typedef [uint8_t sc_pm_reset_type_t](#)
This type is used to declare a desired reset type.
- typedef [uint8_t sc_pm_reset_reason_t](#)
This type is used to declare a reason for a reset.
- typedef [uint8_t sc_pm_sys_if_t](#)
This type is used to specify a system-level interface to be power managed.
- typedef [uint8_t sc_pm_wake_src_t](#)
This type is used to specify a wake source for CPU resources.

Functions

Power Functions

- [sc_err_t sc_pm_set_sys_power_mode](#) ([sc_ipc_t](#) ipc, [sc_pm_power_mode_t](#) mode)
This function sets the system power mode.
- [sc_err_t sc_pm_set_partition_power_mode](#) ([sc_ipc_t](#) ipc, [sc_rm_pt_t](#) pt, [sc_pm_power_mode_t](#) mode)
This function sets the power mode of a partition.
- [sc_err_t sc_pm_get_sys_power_mode](#) ([sc_ipc_t](#) ipc, [sc_rm_pt_t](#) pt, [sc_pm_power_mode_t](#) *mode)
This function gets the power mode of a partition.
- [sc_err_t sc_pm_set_resource_power_mode](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_pm_power_mode_t](#) mode)
This function sets the power mode of a resource.
- [sc_err_t sc_pm_set_resource_power_mode_all](#) ([sc_ipc_t](#) ipc, [sc_rm_pt_t](#) pt, [sc_pm_power_mode_t](#) mode, [sc_rsrc_t](#) exclude)
This function sets the power mode for all the resources owned by a child partition.
- [sc_err_t sc_pm_get_resource_power_mode](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_pm_power_mode_t](#) *mode)
This function gets the power mode of a resource.
- [sc_err_t sc_pm_req_low_power_mode](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_pm_power_mode_t](#) mode)
This function requests the low power mode some of the resources can enter based on their state.
- [sc_err_t sc_pm_req_cpu_low_power_mode](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_pm_power_mode_t](#) mode, [sc_pm_wake_src_t](#) wake_src)
This function requests low-power mode entry for CPU/cluster resources.
- [sc_err_t sc_pm_set_cpu_resume_addr](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_faddr_t](#) address)
This function is used to set the resume address of a CPU.
- [sc_err_t sc_pm_set_cpu_resume](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_bool_t](#) isPrimary, [sc_faddr_t](#) address)
This function is used to set parameters for CPU resume from low-power mode.
- [sc_err_t sc_pm_req_sys_if_power_mode](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_pm_sys_if_t](#) sys_if, [sc_pm_power_mode_t](#) hpm, [sc_pm_power_mode_t](#) lpm)
This function requests the power mode configuration for system-level interfaces including messaging units, interconnect, and memories.

Clock/PLL Functions

- [sc_err_t sc_pm_set_clock_rate](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_pm_clk_t](#) clk, [sc_pm_clock_rate_t](#) *rate)
This function sets the rate of a resource's clock/PLL.
- [sc_err_t sc_pm_get_clock_rate](#) ([sc_ipc_t](#) ipc, [sc_rsrc_t](#) resource, [sc_pm_clk_t](#) clk, [sc_pm_clock_rate_t](#) *rate)
This function gets the rate of a resource's clock/PLL.

- `sc_err_t sc_pm_clock_enable` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_pm_clk_t clk`, `sc_bool_t enable`, `sc_bool_t autog`)
This function enables/disables a resource's clock.
- `sc_err_t sc_pm_set_clock_parent` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_pm_clk_t clk`, `sc_pm_clk_parent_t parent`)
This function sets the parent of a resource's clock.
- `sc_err_t sc_pm_get_clock_parent` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_pm_clk_t clk`, `sc_pm_clk_parent_t *parent`)
This function gets the parent of a resource's clock.

Reset Functions

- `sc_err_t sc_pm_reset` (`sc_ipc_t ipc`, `sc_pm_reset_type_t type`)
This function is used to reset the system.
- `sc_err_t sc_pm_reset_reason` (`sc_ipc_t ipc`, `sc_pm_reset_reason_t *reason`)
This function gets a caller's reset reason.
- `sc_err_t sc_pm_boot` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_rsrc_t resource_cpu`, `sc_faddr_t boot_addr`, `sc_rsrc_t resource_mu`, `sc_rsrc_t resource_dev`)
This function is used to boot a partition.
- `void sc_pm_reboot` (`sc_ipc_t ipc`, `sc_pm_reset_type_t type`)
This function is used to reboot the caller's partition.
- `sc_err_t sc_pm_reboot_partition` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_pm_reset_type_t type`)
This function is used to reboot a partition.
- `sc_err_t sc_pm_cpu_start` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_bool_t enable`, `sc_faddr_t address`)
This function is used to start/stop a CPU.

11.6.1 Detailed Description

Header file containing the public API for the System Controller (SC) Power Management (PM) function.

This includes functions for power state control, clock control, reset control, and wake-up event control.

11.7 platform/svc/rm/api.h File Reference

Header file containing the public API for the System Controller (SC) Resource Management (RM) function.

Macros

Defines for type widths

- `#define SC_RM_PARTITION_W 5U`
Width of `sc_rm_pt_t`.
- `#define SC_RM_MEMREG_W 6U`
Width of `sc_rm_mr_t`.
- `#define SC_RM_DID_W 4U`
Width of `sc_rm_did_t`.
- `#define SC_RM_SID_W 6U`
Width of `sc_rm_sid_t`.

- #define `SC_RM_SPA_W` 2U
Width of `sc_rm_spa_t`.
- #define `SC_RM_PERM_W` 3U
Width of `sc_rm_perm_t`.

Defines for ALL parameters

- #define `SC_RM_PT_ALL` ((`sc_rm_pt_t`) UINT8_MAX)
All partitions.
- #define `SC_RM_MR_ALL` ((`sc_rm_mr_t`) UINT8_MAX)
All memory regions.

Defines for `sc_rm_spa_t`

- #define `SC_RM_SPA_PASSTHRU` 0U
Pass through (attribute driven by master)
- #define `SC_RM_SPA_PASSSID` 1U
Pass through and output on SID.
- #define `SC_RM_SPA_ASSERT` 2U
Assert (force to be secure/privileged)
- #define `SC_RM_SPA_NEGATE` 3U
Negate (force to be non-secure/user)

Defines for `sc_rm_perm_t`

- #define `SC_RM_PERM_NONE` 0U
No access.
- #define `SC_RM_PERM_SEC_R` 1U
Secure RO.
- #define `SC_RM_PERM_SECPRIV_RW` 2U
Secure privilege R/W.
- #define `SC_RM_PERM_SEC_RW` 3U
Secure R/W.
- #define `SC_RM_PERM_NSPRIV_R` 4U
Secure R/W, non-secure privilege RO.
- #define `SC_RM_PERM_NS_R` 5U
Secure R/W, non-secure RO.
- #define `SC_RM_PERM_NSPRIV_RW` 6U
Secure R/W, non-secure privilege R/W.
- #define `SC_RM_PERM_FULL` 7U
Full access.

Typedefs

- typedef `uint8_t sc_rm_pt_t`
This type is used to declare a resource partition.
- typedef `uint8_t sc_rm_mr_t`
This type is used to declare a memory region.
- typedef `uint8_t sc_rm_did_t`
This type is used to declare a resource domain ID used by the isolation HW.
- typedef `uint16_t sc_rm_sid_t`
This type is used to declare an SMMU StreamID.
- typedef `uint8_t sc_rm_spa_t`
This type is a used to declare master transaction attributes.
- typedef `uint8_t sc_rm_perm_t`
This type is used to declare a resource/memory region access permission.

Functions

Partition Functions

- `sc_err_t sc_rm_partition_alloc` (`sc_ipc_t ipc`, `sc_rm_pt_t *pt`, `sc_bool_t secure`, `sc_bool_t isolated`, `sc_bool_t restricted`, `sc_bool_t grant`, `sc_bool_t coherent`)
This function requests that the SC create a new resource partition.
- `sc_err_t sc_rm_set_confidential` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_bool_t retro`)
This function makes a partition confidential.
- `sc_err_t sc_rm_partition_free` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`)
This function frees a partition and assigns all resources to the caller.
- `sc_rm_did_t sc_rm_get_did` (`sc_ipc_t ipc`)
This function returns the DID of a partition.
- `sc_err_t sc_rm_partition_static` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_rm_did_t did`)
This function forces a partition to use a specific static DID.
- `sc_err_t sc_rm_partition_lock` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`)
This function locks a partition.
- `sc_err_t sc_rm_get_partition` (`sc_ipc_t ipc`, `sc_rm_pt_t *pt`)
This function gets the partition handle of the caller.
- `sc_err_t sc_rm_set_parent` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_rm_pt_t pt_parent`)
This function sets a new parent for a partition.
- `sc_err_t sc_rm_move_all` (`sc_ipc_t ipc`, `sc_rm_pt_t pt_src`, `sc_rm_pt_t pt_dst`, `sc_bool_t move_rsrc`, `sc_bool_t move_pads`)
This function moves all movable resources/pads owned by a source partition to a destination partition.

Resource Functions

- `sc_err_t sc_rm_assign_resource` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_rsrc_t resource`)
This function assigns ownership of a resource to a partition.
- `sc_err_t sc_rm_set_resource_movable` (`sc_ipc_t ipc`, `sc_rsrc_t resource_fst`, `sc_rsrc_t resource_lst`, `sc_bool_t movable`)
This function flags resources as movable or not.
- `sc_err_t sc_rm_set_subsys_rsrc_movable` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_bool_t movable`)
This function flags all of a subsystem's resources as movable or not.
- `sc_err_t sc_rm_set_master_attributes` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_rm_spa_t sa`, `sc_rm_spa_t pa`, `sc_bool_t smmu_bypass`)
This function sets attributes for a resource which is a bus master (i.e.
This function sets the StreamID for a resource which is a bus master (i.e.
- `sc_err_t sc_rm_set_master_sid` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_rm_sid_t sid`)
This function sets the StreamID for a resource which is a bus master (i.e.
- `sc_err_t sc_rm_set_peripheral_permissions` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_rm_pt_t pt`, `sc_rm_perm_t perm`)
This function sets access permissions for a peripheral resource.
- `sc_bool_t sc_rm_is_resource_owned` (`sc_ipc_t ipc`, `sc_rsrc_t resource`)
This function gets ownership status of a resource.
- `sc_bool_t sc_rm_is_resource_master` (`sc_ipc_t ipc`, `sc_rsrc_t resource`)
This function is used to test if a resource is a bus master.
- `sc_bool_t sc_rm_is_resource_peripheral` (`sc_ipc_t ipc`, `sc_rsrc_t resource`)
This function is used to test if a resource is a peripheral.
- `sc_err_t sc_rm_get_resource_info` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_rm_sid_t *sid`)
This function is used to obtain info about a resource.

Memory Region Functions

- `sc_err_t sc_rm_memreg_alloc` (`sc_ipc_t ipc`, `sc_rm_mr_t *mr`, `sc_faddr_t addr_start`, `sc_faddr_t addr_end`)

This function requests that the SC create a new memory region.

- `sc_err_t sc_rm_memreg_split` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`, `sc_rm_mr_t *mr_ret`, `sc_faddr_t addr_start`, `sc_faddr_t addr_end`)

This function requests that the SC split a memory region.

- `sc_err_t sc_rm_memreg_free` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`)

This function frees a memory region.

- `sc_err_t sc_rm_find_memreg` (`sc_ipc_t ipc`, `sc_rm_mr_t *mr`, `sc_faddr_t addr_start`, `sc_faddr_t addr_end`)

Internal SC function to find a memory region.

- `sc_err_t sc_rm_assign_memreg` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_rm_mr_t mr`)

This function assigns ownership of a memory region.

- `sc_err_t sc_rm_set_memreg_permissions` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`, `sc_rm_pt_t pt`, `sc_rm_perm_t perm`)

This function sets access permissions for a memory region.

- `sc_bool_t sc_rm_is_memreg_owned` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`)

This function gets ownership status of a memory region.

- `sc_err_t sc_rm_get_memreg_info` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`, `sc_faddr_t *addr_start`, `sc_faddr_t *addr_end`)

This function is used to obtain info about a memory region.

Pad Functions

- `sc_err_t sc_rm_assign_pad` (`sc_ipc_t ipc`, `sc_rm_pt_t pt`, `sc_pad_t pad`)

This function assigns ownership of a pad to a partition.

- `sc_err_t sc_rm_set_pad_movable` (`sc_ipc_t ipc`, `sc_pad_t pad_fst`, `sc_pad_t pad_lst`, `sc_bool_t movable`)

This function flags pads as movable or not.

- `sc_bool_t sc_rm_is_pad_owned` (`sc_ipc_t ipc`, `sc_pad_t pad`)

This function gets ownership status of a pad.

Debug Functions

- `void sc_rm_dump` (`sc_ipc_t ipc`)

This function dumps the RM state for debug.

11.7.1 Detailed Description

Header file containing the public API for the System Controller (SC) Resource Management (RM) function.

This includes functions for partitioning resources, pads, and memory regions.

11.8 platform/svc/timer/api.h File Reference

Header file containing the public API for the System Controller (SC) Timer function.

Macros

Defines for type widths

- `#define SC_TIMER_ACTION_W 3U`
Width of `sc_timer_wdog_action_t`.

Defines for `sc_timer_wdog_action_t`

- `#define SC_TIMER_WDOG_ACTION_PARTITION 0U`
Reset partition.
- `#define SC_TIMER_WDOG_ACTION_WARM 1U`
Warm reset system.
- `#define SC_TIMER_WDOG_ACTION_COLD 2U`
Cold reset system.
- `#define SC_TIMER_WDOG_ACTION_BOARD 3U`
Reset board.
- `#define SC_TIMER_WDOG_ACTION_IRQ 4U`
Only generate IRQs.

Typedefs

- `typedef uint8_t sc_timer_wdog_action_t`
This type is used to configure the watchdog action.
- `typedef uint32_t sc_timer_wdog_time_t`
This type is used to declare a watchdog time value in milliseconds.

Functions

Wathdog Functions

- `sc_err_t sc_timer_set_wdog_timeout (sc_ipc_t ipc, sc_timer_wdog_time_t timeout)`
This function sets the watchdog timeout in milliseconds.
- `sc_err_t sc_timer_set_wdog_pre_timeout (sc_ipc_t ipc, sc_timer_wdog_time_t pre_timeout)`
This function sets the watchdog pre-timeout in milliseconds.
- `sc_err_t sc_timer_start_wdog (sc_ipc_t ipc, sc_bool_t lock)`
This function starts the watchdog.
- `sc_err_t sc_timer_stop_wdog (sc_ipc_t ipc)`
This function stops the watchdog if it is not locked.
- `sc_err_t sc_timer_ping_wdog (sc_ipc_t ipc)`
This function pings (services, kicks) the watchdog resetting the time before expiration back to the timeout.
- `sc_err_t sc_timer_get_wdog_status (sc_ipc_t ipc, sc_timer_wdog_time_t *timeout, sc_timer_wdog_time_t *max_timeout, sc_timer_wdog_time_t *remaining_time)`
This function gets the status of the watchdog.
- `sc_err_t sc_timer_pt_get_wdog_status (sc_ipc_t ipc, sc_rm_pt_t pt, sc_bool_t *enb, sc_timer_wdog_time_t *timeout, sc_timer_wdog_time_t *remaining_time)`
This function gets the status of the watchdog of a partition.
- `sc_err_t sc_timer_set_wdog_action (sc_ipc_t ipc, sc_rm_pt_t pt, sc_timer_wdog_action_t action)`
This function configures the action to be taken when a watchdog expires.

Real-Time Clock (RTC) Functions

- [sc_err_t sc_timer_set_rtc_time](#) (sc_ipc_t ipc, [uint16_t](#) year, [uint8_t](#) mon, [uint8_t](#) day, [uint8_t](#) hour, [uint8_t](#) min, [uint8_t](#) sec)
This function sets the RTC time.
- [sc_err_t sc_timer_get_rtc_time](#) (sc_ipc_t ipc, [uint16_t](#) *year, [uint8_t](#) *mon, [uint8_t](#) *day, [uint8_t](#) *hour, [uint8_t](#) *min, [uint8_t](#) *sec)
This function gets the RTC time.
- [sc_err_t sc_timer_get_rtc_sec1970](#) (sc_ipc_t ipc, [uint32_t](#) *sec)
This function gets the RTC time in seconds since 1/1/1970.
- [sc_err_t sc_timer_set_rtc_alarm](#) (sc_ipc_t ipc, [uint16_t](#) year, [uint8_t](#) mon, [uint8_t](#) day, [uint8_t](#) hour, [uint8_t](#) min, [uint8_t](#) sec)
This function sets the RTC alarm.
- [sc_err_t sc_timer_set_rtc_periodic_alarm](#) (sc_ipc_t ipc, [uint32_t](#) sec)
This function sets the RTC alarm (periodic mode).
- [sc_err_t sc_timer_cancel_rtc_alarm](#) (sc_ipc_t ipc)
This function cancels the RTC alarm.
- [sc_err_t sc_timer_set_rtc_calb](#) (sc_ipc_t ipc, [int8_t](#) count)
This function sets the RTC calibration value.

System Counter (SYSCTR) Functions

- [sc_err_t sc_timer_set_sysctr_alarm](#) (sc_ipc_t ipc, [uint64_t](#) ticks)
This function sets the SYSCTR alarm.
- [sc_err_t sc_timer_set_sysctr_periodic_alarm](#) (sc_ipc_t ipc, [uint64_t](#) ticks)
This function sets the SYSCTR alarm (periodic mode).
- [sc_err_t sc_timer_cancel_sysctr_alarm](#) (sc_ipc_t ipc)
This function cancels the SYSCTR alarm.

11.8.1 Detailed Description

Header file containing the public API for the System Controller (SC) Timer function.

Index

- (SVC) Interrupt Service, 45
 - sc_irq_enable, 47
 - sc_irq_status, 48
- (SVC) Miscellaneous Service, 50
 - sc_misc_boot_done, 67
 - sc_misc_boot_status, 66
 - sc_misc_build_info, 65
 - sc_misc_debug_out, 64
 - sc_misc_get_boot_dev, 70
 - sc_misc_get_boot_type, 70
 - sc_misc_get_button_status, 71
 - sc_misc_get_control, 54
 - sc_misc_get_temp, 69
 - sc_misc_otf_fuse_read, 67
 - sc_misc_otf_fuse_write, 68
 - sc_misc_rompatch_checksum, 71
 - sc_misc_seco_attest, 61
 - sc_misc_seco_attest_mode, 60
 - sc_misc_seco_attest_verify, 63
 - sc_misc_seco_authenticate, 56
 - sc_misc_seco_build_info, 59
 - sc_misc_seco_chip_info, 60
 - sc_misc_seco_commit, 63
 - sc_misc_seco_enable_debug, 58
 - sc_misc_seco_forward_lifecycle, 58
 - sc_misc_seco_fuse_write, 57
 - sc_misc_seco_get_attest_pkey, 61
 - sc_misc_seco_get_attest_sign, 62
 - sc_misc_seco_image_load, 56
 - sc_misc_seco_return_lifecycle, 59
 - sc_misc_set_ari, 66
 - sc_misc_set_control, 53
 - sc_misc_set_dma_group, 55
 - sc_misc_set_max_dma_group, 54
 - sc_misc_set_temp, 68
 - sc_misc_unique_id, 65
 - sc_misc_waveform_capture, 64
- (SVC) Pad Service, 72
 - sc_pad_28fdsoi_dse_t, 77
 - sc_pad_28fdsoi_ps_t, 77
 - sc_pad_28fdsoi_pus_t, 77
 - sc_pad_config_t, 77
 - sc_pad_get, 83
 - sc_pad_get_all, 82
 - sc_pad_get_gp, 79
 - sc_pad_get_gp_28fdsoi, 85
 - sc_pad_get_gp_28fdsoi_comp, 88
 - sc_pad_get_gp_28fdsoi_hsic, 86
 - sc_pad_get_mux, 78
 - sc_pad_get_wakeup, 81
 - sc_pad_iso_t, 77
 - sc_pad_set, 83
 - sc_pad_set_all, 81
 - sc_pad_set_gp, 79
 - sc_pad_set_gp_28fdsoi, 84
 - sc_pad_set_gp_28fdsoi_comp, 87
 - sc_pad_set_gp_28fdsoi_hsic, 85
 - sc_pad_set_mux, 78
 - sc_pad_set_wakeup, 80
- (SVC) Power Management Service, 90
 - SC_PM_CLK_MODE_ON, 96
 - SC_PM_CLK_MODE_ROM_INIT, 96
 - SC_PM_PARENT_BYPS, 97
 - SC_PM_PARENT_XTAL, 96
 - sc_pm_boot, 109
 - sc_pm_clock_enable, 105
 - sc_pm_cpu_start, 110
 - sc_pm_get_clock_parent, 107
 - sc_pm_get_clock_rate, 105
 - sc_pm_get_resource_power_mode, 101
 - sc_pm_get_sys_power_mode, 99
 - sc_pm_power_mode_t, 97
 - sc_pm_reboot, 109
 - sc_pm_reboot_partition, 110
 - sc_pm_req_cpu_low_power_mode, 102
 - sc_pm_req_low_power_mode, 101
 - sc_pm_req_sys_if_power_mode, 103
 - sc_pm_reset, 107
 - sc_pm_reset_reason, 108
 - sc_pm_set_clock_parent, 106
 - sc_pm_set_clock_rate, 104
 - sc_pm_set_cpu_resume, 103
 - sc_pm_set_cpu_resume_addr, 102
 - sc_pm_set_partition_power_mode, 98
 - sc_pm_set_resource_power_mode, 99
 - sc_pm_set_resource_power_mode_all, 100
 - sc_pm_set_sys_power_mode, 97
- (SVC) Resource Management Service, 112
 - sc_rm_assign_memreg, 133
 - sc_rm_assign_pad, 135

- sc_rm_assign_resource, 124
- sc_rm_dump, 137
- sc_rm_find_memreg, 132
- sc_rm_get_did, 120
- sc_rm_get_memreg_info, 135
- sc_rm_get_partition, 122
- sc_rm_get_resource_info, 130
- sc_rm_is_memreg_owned, 134
- sc_rm_is_pad_owned, 136
- sc_rm_is_resource_master, 129
- sc_rm_is_resource_owned, 128
- sc_rm_is_resource_peripheral, 129
- sc_rm_memreg_alloc, 130
- sc_rm_memreg_free, 132
- sc_rm_memreg_split, 131
- sc_rm_move_all, 123
- sc_rm_partition_alloc, 118
- sc_rm_partition_free, 119
- sc_rm_partition_lock, 122
- sc_rm_partition_static, 120
- sc_rm_perm_t, 117
- sc_rm_set_confidential, 119
- sc_rm_set_master_attributes, 126
- sc_rm_set_master_sid, 127
- sc_rm_set_memreg_permissions, 133
- sc_rm_set_pad_movable, 136
- sc_rm_set_parent, 123
- sc_rm_set_peripheral_permissions, 128
- sc_rm_set_resource_movable, 125
- sc_rm_set_subsys_rsrc_movable, 126
- (SVC) Timer Service, 138
 - sc_timer_cancel_rtc_alarm, 146
 - sc_timer_cancel_sysctr_alarm, 148
 - sc_timer_get_rtc_sec1970, 145
 - sc_timer_get_rtc_time, 144
 - sc_timer_get_wdog_status, 141
 - sc_timer_ping_wdog, 141
 - sc_timer_pt_get_wdog_status, 142
 - sc_timer_set_rtc_alarm, 145
 - sc_timer_set_rtc_calb, 147
 - sc_timer_set_rtc_periodic_alarm, 146
 - sc_timer_set_rtc_time, 143
 - sc_timer_set_sysctr_alarm, 147
 - sc_timer_set_sysctr_periodic_alarm, 148
 - sc_timer_set_wdog_action, 143
 - sc_timer_set_wdog_pre_timeout, 140
 - sc_timer_set_wdog_timeout, 139
 - sc_timer_start_wdog, 140
 - sc_timer_stop_wdog, 141
- ipc.h
 - sc_ipc_close, 152
 - sc_ipc_open, 151
 - sc_ipc_read, 152
 - sc_ipc_write, 152
- platform/main/ipc.h, 151
- platform/main/types.h, 154
- platform/svc/irq/api.h, 170
- platform/svc/misc/api.h, 173
- platform/svc/pad/api.h, 176
- platform/svc/pm/api.h, 179
- platform/svc/rm/api.h, 184
- platform/svc/timer/api.h, 187
- SC_PM_CLK_MODE_ON
 - (SVC) Power Management Service, 96
- SC_PM_CLK_MODE_ROM_INIT
 - (SVC) Power Management Service, 96
- SC_PM_PARENT_BYPS
 - (SVC) Power Management Service, 97
- SC_PM_PARENT_XTAL
 - (SVC) Power Management Service, 96
- sc_ipc_close
 - ipc.h, 152
- sc_ipc_open
 - ipc.h, 151
- sc_ipc_read
 - ipc.h, 152
- sc_ipc_write
 - ipc.h, 152
- sc_irq_enable
 - (SVC) Interrupt Service, 47
- sc_irq_status
 - (SVC) Interrupt Service, 48
- sc_misc_boot_done
 - (SVC) Miscellaneous Service, 67
- sc_misc_boot_status
 - (SVC) Miscellaneous Service, 66
- sc_misc_build_info
 - (SVC) Miscellaneous Service, 65
- sc_misc_debug_out
 - (SVC) Miscellaneous Service, 64
- sc_misc_get_boot_dev
 - (SVC) Miscellaneous Service, 70
- sc_misc_get_boot_type
 - (SVC) Miscellaneous Service, 70
- sc_misc_get_button_status
 - (SVC) Miscellaneous Service, 71
- sc_misc_get_control
 - (SVC) Miscellaneous Service, 54
- sc_misc_get_temp
 - (SVC) Miscellaneous Service, 69
- sc_misc_otp_fuse_read
 - (SVC) Miscellaneous Service, 67
- sc_misc_otp_fuse_write
 - (SVC) Miscellaneous Service, 68
- sc_misc_rompatch_checksum
 - (SVC) Miscellaneous Service, 71

- sc_misc_seco_attest
 - (SVC) Miscellaneous Service, [61](#)
- sc_misc_seco_attest_mode
 - (SVC) Miscellaneous Service, [60](#)
- sc_misc_seco_attest_verify
 - (SVC) Miscellaneous Service, [63](#)
- sc_misc_seco_authenticate
 - (SVC) Miscellaneous Service, [56](#)
- sc_misc_seco_build_info
 - (SVC) Miscellaneous Service, [59](#)
- sc_misc_seco_chip_info
 - (SVC) Miscellaneous Service, [60](#)
- sc_misc_seco_commit
 - (SVC) Miscellaneous Service, [63](#)
- sc_misc_seco_enable_debug
 - (SVC) Miscellaneous Service, [58](#)
- sc_misc_seco_forward_lifecycle
 - (SVC) Miscellaneous Service, [58](#)
- sc_misc_seco_fuse_write
 - (SVC) Miscellaneous Service, [57](#)
- sc_misc_seco_get_attest_pkey
 - (SVC) Miscellaneous Service, [61](#)
- sc_misc_seco_get_attest_sign
 - (SVC) Miscellaneous Service, [62](#)
- sc_misc_seco_image_load
 - (SVC) Miscellaneous Service, [56](#)
- sc_misc_seco_return_lifecycle
 - (SVC) Miscellaneous Service, [59](#)
- sc_misc_set_ari
 - (SVC) Miscellaneous Service, [66](#)
- sc_misc_set_control
 - (SVC) Miscellaneous Service, [53](#)
- sc_misc_set_dma_group
 - (SVC) Miscellaneous Service, [55](#)
- sc_misc_set_max_dma_group
 - (SVC) Miscellaneous Service, [54](#)
- sc_misc_set_temp
 - (SVC) Miscellaneous Service, [68](#)
- sc_misc_unique_id
 - (SVC) Miscellaneous Service, [65](#)
- sc_misc_waveform_capture
 - (SVC) Miscellaneous Service, [64](#)
- sc_pad_28fdsoi_dse_t
 - (SVC) Pad Service, [77](#)
- sc_pad_28fdsoi_ps_t
 - (SVC) Pad Service, [77](#)
- sc_pad_28fdsoi_pus_t
 - (SVC) Pad Service, [77](#)
- sc_pad_config_t
 - (SVC) Pad Service, [77](#)
- sc_pad_get
 - (SVC) Pad Service, [83](#)
- sc_pad_get_all
 - (SVC) Pad Service, [82](#)
- sc_pad_get_gp
 - (SVC) Pad Service, [79](#)
- sc_pad_get_gp_28fdsoi
 - (SVC) Pad Service, [85](#)
- sc_pad_get_gp_28fdsoi_comp
 - (SVC) Pad Service, [88](#)
- sc_pad_get_gp_28fdsoi_hsic
 - (SVC) Pad Service, [86](#)
- sc_pad_get_mux
 - (SVC) Pad Service, [78](#)
- sc_pad_get_wakeup
 - (SVC) Pad Service, [81](#)
- sc_pad_iso_t
 - (SVC) Pad Service, [77](#)
- sc_pad_set
 - (SVC) Pad Service, [83](#)
- sc_pad_set_all
 - (SVC) Pad Service, [81](#)
- sc_pad_set_gp
 - (SVC) Pad Service, [79](#)
- sc_pad_set_gp_28fdsoi
 - (SVC) Pad Service, [84](#)
- sc_pad_set_gp_28fdsoi_comp
 - (SVC) Pad Service, [87](#)
- sc_pad_set_gp_28fdsoi_hsic
 - (SVC) Pad Service, [85](#)
- sc_pad_set_mux
 - (SVC) Pad Service, [78](#)
- sc_pad_set_wakeup
 - (SVC) Pad Service, [80](#)
- sc_pad_t
 - types.h, [170](#)
- sc_pm_boot
 - (SVC) Power Management Service, [109](#)
- sc_pm_clock_enable
 - (SVC) Power Management Service, [105](#)
- sc_pm_cpu_start
 - (SVC) Power Management Service, [110](#)
- sc_pm_get_clock_parent
 - (SVC) Power Management Service, [107](#)
- sc_pm_get_clock_rate
 - (SVC) Power Management Service, [105](#)
- sc_pm_get_resource_power_mode
 - (SVC) Power Management Service, [101](#)
- sc_pm_get_sys_power_mode
 - (SVC) Power Management Service, [99](#)
- sc_pm_power_mode_t
 - (SVC) Power Management Service, [97](#)
- sc_pm_reboot
 - (SVC) Power Management Service, [109](#)
- sc_pm_reboot_partition
 - (SVC) Power Management Service, [110](#)
- sc_pm_req_cpu_low_power_mode
 - (SVC) Power Management Service, [102](#)

- sc_pm_req_low_power_mode
(SVC) Power Management Service, [101](#)
- sc_pm_req_sys_if_power_mode
(SVC) Power Management Service, [103](#)
- sc_pm_reset
(SVC) Power Management Service, [107](#)
- sc_pm_reset_reason
(SVC) Power Management Service, [108](#)
- sc_pm_set_clock_parent
(SVC) Power Management Service, [106](#)
- sc_pm_set_clock_rate
(SVC) Power Management Service, [104](#)
- sc_pm_set_cpu_resume
(SVC) Power Management Service, [103](#)
- sc_pm_set_cpu_resume_addr
(SVC) Power Management Service, [102](#)
- sc_pm_set_partition_power_mode
(SVC) Power Management Service, [98](#)
- sc_pm_set_resource_power_mode
(SVC) Power Management Service, [99](#)
- sc_pm_set_resource_power_mode_all
(SVC) Power Management Service, [100](#)
- sc_pm_set_sys_power_mode
(SVC) Power Management Service, [97](#)
- sc_rm_assign_memreg
(SVC) Resource Management Service, [133](#)
- sc_rm_assign_pad
(SVC) Resource Management Service, [135](#)
- sc_rm_assign_resource
(SVC) Resource Management Service, [124](#)
- sc_rm_dump
(SVC) Resource Management Service, [137](#)
- sc_rm_find_memreg
(SVC) Resource Management Service, [132](#)
- sc_rm_get_did
(SVC) Resource Management Service, [120](#)
- sc_rm_get_memreg_info
(SVC) Resource Management Service, [135](#)
- sc_rm_get_partition
(SVC) Resource Management Service, [122](#)
- sc_rm_get_resource_info
(SVC) Resource Management Service, [130](#)
- sc_rm_is_memreg_owned
(SVC) Resource Management Service, [134](#)
- sc_rm_is_pad_owned
(SVC) Resource Management Service, [136](#)
- sc_rm_is_resource_master
(SVC) Resource Management Service, [129](#)
- sc_rm_is_resource_owned
(SVC) Resource Management Service, [128](#)
- sc_rm_is_resource_peripheral
(SVC) Resource Management Service, [129](#)
- sc_rm_memreg_alloc
(SVC) Resource Management Service, [130](#)
- sc_rm_memreg_free
(SVC) Resource Management Service, [132](#)
- sc_rm_memreg_split
(SVC) Resource Management Service, [131](#)
- sc_rm_move_all
(SVC) Resource Management Service, [123](#)
- sc_rm_partition_alloc
(SVC) Resource Management Service, [118](#)
- sc_rm_partition_free
(SVC) Resource Management Service, [119](#)
- sc_rm_partition_lock
(SVC) Resource Management Service, [122](#)
- sc_rm_partition_static
(SVC) Resource Management Service, [120](#)
- sc_rm_perm_t
(SVC) Resource Management Service, [117](#)
- sc_rm_set_confidential
(SVC) Resource Management Service, [119](#)
- sc_rm_set_master_attributes
(SVC) Resource Management Service, [126](#)
- sc_rm_set_master_sid
(SVC) Resource Management Service, [127](#)
- sc_rm_set_memreg_permissions
(SVC) Resource Management Service, [133](#)
- sc_rm_set_pad_movable
(SVC) Resource Management Service, [136](#)
- sc_rm_set_parent
(SVC) Resource Management Service, [123](#)
- sc_rm_set_peripheral_permissions
(SVC) Resource Management Service, [128](#)
- sc_rm_set_resource_movable
(SVC) Resource Management Service, [125](#)
- sc_rm_set_subsys_rsrc_movable
(SVC) Resource Management Service, [126](#)
- sc_rsrc_t
types.h, [170](#)
- sc_timer_cancel_rtc_alarm
(SVC) Timer Service, [146](#)
- sc_timer_cancel_sysctr_alarm
(SVC) Timer Service, [148](#)
- sc_timer_get_rtc_sec1970
(SVC) Timer Service, [145](#)
- sc_timer_get_rtc_time
(SVC) Timer Service, [144](#)
- sc_timer_get_wdog_status
(SVC) Timer Service, [141](#)
- sc_timer_ping_wdog
(SVC) Timer Service, [141](#)
- sc_timer_pt_get_wdog_status
(SVC) Timer Service, [142](#)
- sc_timer_set_rtc_alarm
(SVC) Timer Service, [145](#)
- sc_timer_set_rtc_calb
(SVC) Timer Service, [147](#)

sc_timer_set_rtc_periodic_alarm
 (SVC) Timer Service, [146](#)
sc_timer_set_rtc_time
 (SVC) Timer Service, [143](#)
sc_timer_set_sysctr_alarm
 (SVC) Timer Service, [147](#)
sc_timer_set_sysctr_periodic_alarm
 (SVC) Timer Service, [148](#)
sc_timer_set_wdog_action
 (SVC) Timer Service, [143](#)
sc_timer_set_wdog_pre_timeout
 (SVC) Timer Service, [140](#)
sc_timer_set_wdog_timeout
 (SVC) Timer Service, [139](#)
sc_timer_start_wdog
 (SVC) Timer Service, [140](#)
sc_timer_stop_wdog
 (SVC) Timer Service, [141](#)

types.h
 sc_pad_t, [170](#)
 sc_rsrc_t, [170](#)