

System Controller Firmware API Reference Guide

i.MX8QXP Die (Ver 1.21)

NXP

Fri Jun 19 2020 19:02:22

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 Security Service	4
1.3.7 Miscellaneous Service	4
2 Disclaimer	5
3 Usage	7
3.1 SCFW API	7
3.2 Loading	7
3.3 Boot Flags	7
3.4 CPU Start Address	8
3.4.1 Cortex-A Start Address	8
3.4.2 Cortex-M Start Address	9
3.5 Cortex-M4 DDR Aliasing	9
4 Resource Management	11
4.1 Partitions	11
4.2 Resources	12
4.3 Pads	13
4.4 Memory Regions	13
4.5 SCFW API	14
4.6 Hardware Resource/Memory Isolation	14
4.7 Example	16
5 Power Management	19
5.1 Wake from Low Power	19
5.1.1 Wake Event Sources	19
5.1.2 GIC Wake Events	19
5.1.3 IRQSTEER Wake Events	20
5.1.4 SCU Wake Events	20
5.1.5 Pad Wake Events	21
5.2 System Power Off	21
5.3 Reset Control	21

6 Resource List	23
7 Clock List	33
8 Control List	37
9 Pad List	41
10 RPC Protocol	49
10.1 Remote Procedure Call	49
10.2 Inter-Processor Communication	50
10.3 Interrupts	50
10.3.1 SCFW to Client Interrupts	50
10.3.1.1 Interrupt Service Request	50
10.3.1.2 Cortex-M Wake	50
10.3.1.3 Cold Boot Flag	51
10.3.1.4 Cortex-M Debug Wake	51
10.3.2 Client to SCFW Interrupts	51
10.3.2.1 RPC/IPC Reset Request	51
10.4 Flags/Status	51
10.4.1 SCFW to Client Flags/Status	51
10.5 Porting	52
10.6 API Message Formats	52
10.6.1 sc_pm_set_sys_power_mode()	52
10.6.2 sc_pm_set_partition_power_mode()	53
10.6.3 sc_pm_get_sys_power_mode()	53
10.6.4 sc_pm_partition_wake()	53
10.6.5 sc_pm_set_resource_power_mode()	53
10.6.6 sc_pm_set_resource_power_mode_all()	54
10.6.7 sc_pm_get_resource_power_mode()	54
10.6.8 sc_pm_req_low_power_mode()	54
10.6.9 sc_pm_req_cpu_low_power_mode()	54
10.6.10 sc_pm_set_cpu_resume_addr()	55
10.6.11 sc_pm_set_cpu_resume()	55
10.6.12 sc_pm_req_sys_if_power_mode()	55
10.6.13 sc_pm_set_clock_rate()	55
10.6.14 sc_pm_get_clock_rate()	56
10.6.15 sc_pm_clock_enable()	56
10.6.16 sc_pm_set_clock_parent()	56
10.6.17 sc_pm_get_clock_parent()	57
10.6.18 sc_pm_reset()	57

10.6.19 sc_pm_reset_reason()	57
10.6.20 sc_pm_get_reset_part()	57
10.6.21 sc_pm_boot()	58
10.6.22 sc_pm_set_boot_parm()	58
10.6.23 sc_pm_reboot()	58
10.6.24 sc_pm_reboot_partition()	58
10.6.25 sc_pm_reboot_continue()	59
10.6.26 sc_pm_cpu_start()	59
10.6.27 sc_pm_cpu_reset()	59
10.6.28 sc_pm_resource_reset()	59
10.6.29 sc_pm_is_partition_started()	60
10.6.30 sc_rm_partition_alloc()	60
10.6.31 sc_rm_set_confidential()	60
10.6.32 sc_rm_partition_free()	60
10.6.33 sc_rm_get_did()	61
10.6.34 sc_rm_partition_static()	61
10.6.35 sc_rm_partition_lock()	61
10.6.36 sc_rm_get_partition()	61
10.6.37 sc_rm_set_parent()	62
10.6.38 sc_rm_move_all()	62
10.6.39 sc_rm_assign_resource()	62
10.6.40 sc_rm_set_resource_movable()	62
10.6.41 sc_rm_set_subsys_rsrc_movable()	63
10.6.42 sc_rm_set_master_attributes()	63
10.6.43 sc_rm_set_master_sid()	63
10.6.44 sc_rm_set_peripheral_permissions()	63
10.6.45 sc_rm_is_resource_owned()	64
10.6.46 sc_rm_get_resource_owner()	64
10.6.47 sc_rm_is_resource_master()	64
10.6.48 sc_rm_is_resource_peripheral()	64
10.6.49 sc_rm_get_resource_info()	65
10.6.50 sc_rm_memreg_alloc()	65
10.6.51 sc_rm_memreg_split()	65
10.6.52 sc_rm_memreg_frag()	66
10.6.53 sc_rm_memreg_free()	66
10.6.54 sc_rm_find_memreg()	66
10.6.55 sc_rm_assign_memreg()	67
10.6.56 sc_rm_set_memreg_permissions()	67
10.6.57 sc_rm_set_memreg_iee()	67

10.6.58	sc_rm_is_memreg_owned()	67
10.6.59	sc_rm_get_memreg_info()	68
10.6.60	sc_rm_assign_pad()	68
10.6.61	sc_rm_set_pad_movable()	68
10.6.62	sc_rm_is_pad_owned()	68
10.6.63	sc_rm_dump()	69
10.6.64	sc_timer_set_wdog_timeout()	69
10.6.65	sc_timer_set_wdog_pre_timeout()	69
10.6.66	sc_timer_set_wdog_window()	69
10.6.67	sc_timer_start_wdog()	70
10.6.68	sc_timer_stop_wdog()	70
10.6.69	sc_timer_ping_wdog()	70
10.6.70	sc_timer_get_wdog_status()	70
10.6.71	sc_timer_pt_get_wdog_status()	71
10.6.72	sc_timer_set_wdog_action()	71
10.6.73	sc_timer_set_rtc_time()	71
10.6.74	sc_timer_get_rtc_time()	71
10.6.75	sc_timer_get_rtc_sec1970()	72
10.6.76	sc_timer_set_rtc_alarm()	72
10.6.77	sc_timer_set_rtc_periodic_alarm()	72
10.6.78	sc_timer_cancel_rtc_alarm()	73
10.6.79	sc_timer_set_rtc_calb()	73
10.6.80	sc_timer_set_sysctr_alarm()	73
10.6.81	sc_timer_set_sysctr_periodic_alarm()	73
10.6.82	sc_timer_cancel_sysctr_alarm()	74
10.6.83	sc_pad_set_mux()	74
10.6.84	sc_pad_get_mux()	74
10.6.85	sc_pad_set_gp()	74
10.6.86	sc_pad_get_gp()	75
10.6.87	sc_pad_set_wakeup()	75
10.6.88	sc_pad_get_wakeup()	75
10.6.89	sc_pad_set_all()	75
10.6.90	sc_pad_get_all()	76
10.6.91	sc_pad_set()	76
10.6.92	sc_pad_get()	76
10.6.93	sc_pad_config()	76
10.6.94	sc_pad_set_gp_28fdsoi()	77
10.6.95	sc_pad_get_gp_28fdsoi()	77
10.6.96	sc_pad_set_gp_28fdsoi_hsic()	77

10.6.97	sc_pad_get_gp_28fdsoi_hsic()	78
10.6.98	sc_pad_set_gp_28fdsoi_comp()	78
10.6.99	sc_pad_get_gp_28fdsoi_comp()	78
10.6.100	sc_misc_set_control()	78
10.6.101	sc_misc_get_control()	79
10.6.102	sc_misc_set_max_dma_group()	79
10.6.103	sc_misc_set_dma_group()	79
10.6.104	sc_misc_debug_out()	80
10.6.105	sc_misc_waveform_capture()	80
10.6.106	sc_misc_build_info()	80
10.6.107	sc_misc_api_ver()	80
10.6.108	sc_misc_unique_id()	81
10.6.109	sc_misc_set_ari()	81
10.6.110	sc_misc_boot_status()	81
10.6.111	sc_misc_boot_done()	81
10.6.112	sc_misc_otp_fuse_read()	82
10.6.113	sc_misc_otp_fuse_write()	82
10.6.114	sc_misc_set_temp()	82
10.6.115	sc_misc_get_temp()	82
10.6.116	sc_misc_get_boot_dev()	83
10.6.117	sc_misc_get_boot_type()	83
10.6.118	sc_misc_get_boot_container()	83
10.6.119	sc_misc_get_button_status()	83
10.6.120	sc_misc_rompatch_checksum()	84
10.6.121	sc_misc_board_ioctl()	84
10.6.122	sc_seco_image_load()	84
10.6.123	sc_seco_authenticate()	85
10.6.124	sc_seco_enh_authenticate()	85
10.6.125	sc_seco_forward_lifecycle()	85
10.6.126	sc_seco_return_lifecycle()	86
10.6.127	sc_seco_commit()	86
10.6.128	sc_seco_attest_mode()	86
10.6.129	sc_seco_attest()	86
10.6.130	sc_seco_get_attest_pkey()	87
10.6.131	sc_seco_get_attest_sign()	87
10.6.132	sc_seco_attest_verify()	87
10.6.133	sc_seco_gen_key_blob()	87
10.6.134	sc_seco_load_key()	88
10.6.135	sc_seco_get_mp_key()	88

10.6.136	sc_seco_update_mpmr()	88
10.6.137	sc_seco_get_mp_sign()	89
10.6.138	sc_seco_build_info()	89
10.6.139	sc_seco_chip_info()	89
10.6.140	sc_seco_enable_debug()	90
10.6.141	sc_seco_get_event()	90
10.6.142	sc_seco_fuse_write()	90
10.6.143	sc_seco_patch()	91
10.6.144	sc_seco_set_mono_counter_partition()	91
10.6.145	sc_seco_set_fips_mode()	91
10.6.146	sc_seco_start_rng()	91
10.6.147	sc_seco_sab_msg()	92
10.6.148	sc_seco_secvio_enable()	92
10.6.149	sc_seco_secvio_config()	92
10.6.150	sc_seco_secvio_dgo_config()	93
10.6.151	sc_irq_enable()	93
10.6.152	sc_irq_status()	93
11	Module Index	95
11.1	Modules	95
12	File Index	97
12.1	File List	97
13	Module Documentation	99
13.1	IRQ: Interrupt Service	99
13.1.1	Detailed Description	102
13.1.2	Function Documentation	102
13.1.2.1	sc_irq_enable()	102
13.1.2.2	sc_irq_status()	103
13.2	MISC: Miscellaneous Service	104
13.2.1	Detailed Description	106
13.2.2	Function Documentation	106
13.2.2.1	sc_misc_set_control()	106
13.2.2.2	sc_misc_get_control()	107
13.2.2.3	sc_misc_set_max_dma_group()	108
13.2.2.4	sc_misc_set_dma_group()	108
13.2.2.5	sc_misc_debug_out()	109
13.2.2.6	sc_misc_waveform_capture()	109
13.2.2.7	sc_misc_build_info()	110

13.2.2.8	sc_misc_api_ver()	110
13.2.2.9	sc_misc_unique_id()	110
13.2.2.10	sc_misc_set_ari()	111
13.2.2.11	sc_misc_boot_status()	111
13.2.2.12	sc_misc_boot_done()	112
13.2.2.13	sc_misc_otp_fuse_read()	112
13.2.2.14	sc_misc_otp_fuse_write()	113
13.2.2.15	sc_misc_set_temp()	114
13.2.2.16	sc_misc_get_temp()	114
13.2.2.17	sc_misc_get_boot_dev()	115
13.2.2.18	sc_misc_get_boot_type()	115
13.2.2.19	sc_misc_get_boot_container()	116
13.2.2.20	sc_misc_get_button_status()	116
13.2.2.21	sc_misc_rompatch_checksum()	117
13.2.2.22	sc_misc_board_ioctl()	117
13.3	PAD: Pad Service	118
13.3.1	Detailed Description	121
13.3.2	Typedef Documentation	123
13.3.2.1	sc_pad_config_t	123
13.3.2.2	sc_pad_iso_t	123
13.3.2.3	sc_pad_28fdsoi_dse_t	123
13.3.2.4	sc_pad_28fdsoi_ps_t	123
13.3.2.5	sc_pad_28fdsoi_pus_t	123
13.3.3	Function Documentation	123
13.3.3.1	sc_pad_set_mux()	123
13.3.3.2	sc_pad_get_mux()	124
13.3.3.3	sc_pad_set_gp()	125
13.3.3.4	sc_pad_get_gp()	125
13.3.3.5	sc_pad_set_wakeup()	126
13.3.3.6	sc_pad_get_wakeup()	127
13.3.3.7	sc_pad_set_all()	127
13.3.3.8	sc_pad_get_all()	128
13.3.3.9	sc_pad_set()	129
13.3.3.10	sc_pad_get()	129
13.3.3.11	sc_pad_config()	130
13.3.3.12	sc_pad_set_gp_28fdsoi()	131
13.3.3.13	sc_pad_get_gp_28fdsoi()	131
13.3.3.14	sc_pad_set_gp_28fdsoi_hsic()	132
13.3.3.15	sc_pad_get_gp_28fdsoi_hsic()	133

13.3.3.16	sc_pad_set_gp_28fdsoi_comp()	133
13.3.3.17	sc_pad_get_gp_28fdsoi_comp()	134
13.4	PM: Power Management Service	136
13.4.1	Detailed Description	141
13.4.2	Typedef Documentation	141
13.4.2.1	sc_pm_power_mode_t	142
13.4.3	Function Documentation	142
13.4.3.1	sc_pm_set_sys_power_mode()	142
13.4.3.2	sc_pm_set_partition_power_mode()	142
13.4.3.3	sc_pm_get_sys_power_mode()	143
13.4.3.4	sc_pm_partition_wake()	144
13.4.3.5	sc_pm_set_resource_power_mode()	144
13.4.3.6	sc_pm_set_resource_power_mode_all()	145
13.4.3.7	sc_pm_get_resource_power_mode()	146
13.4.3.8	sc_pm_req_low_power_mode()	146
13.4.3.9	sc_pm_req_cpu_low_power_mode()	147
13.4.3.10	sc_pm_set_cpu_resume_addr()	147
13.4.3.11	sc_pm_set_cpu_resume()	148
13.4.3.12	sc_pm_req_sys_if_power_mode()	148
13.4.3.13	sc_pm_set_clock_rate()	149
13.4.3.14	sc_pm_get_clock_rate()	150
13.4.3.15	sc_pm_clock_enable()	150
13.4.3.16	sc_pm_set_clock_parent()	151
13.4.3.17	sc_pm_get_clock_parent()	152
13.4.3.18	sc_pm_reset()	152
13.4.3.19	sc_pm_reset_reason()	153
13.4.3.20	sc_pm_get_reset_part()	153
13.4.3.21	sc_pm_boot()	154
13.4.3.22	sc_pm_set_boot_parm()	155
13.4.3.23	sc_pm_reboot()	155
13.4.3.24	sc_pm_reboot_partition()	156
13.4.3.25	sc_pm_reboot_continue()	156
13.4.3.26	sc_pm_cpu_start()	157
13.4.3.27	sc_pm_cpu_reset()	158
13.4.3.28	sc_pm_resource_reset()	158
13.4.3.29	sc_pm_is_partition_started()	159
13.5	RM: Resource Management Service	160
13.5.1	Detailed Description	163
13.5.2	Typedef Documentation	166

13.5.2.1	sc_rm_perm_t	166
13.5.2.2	sc_rm_rmsg_t	166
13.5.3	Function Documentation	166
13.5.3.1	sc_rm_partition_alloc()	166
13.5.3.2	sc_rm_set_confidential()	167
13.5.3.3	sc_rm_partition_free()	168
13.5.3.4	sc_rm_get_did()	168
13.5.3.5	sc_rm_partition_static()	169
13.5.3.6	sc_rm_partition_lock()	169
13.5.3.7	sc_rm_get_partition()	170
13.5.3.8	sc_rm_set_parent()	170
13.5.3.9	sc_rm_move_all()	171
13.5.3.10	sc_rm_assign_resource()	172
13.5.3.11	sc_rm_set_resource_movable()	173
13.5.3.12	sc_rm_set_subsys_rsrc_movable()	173
13.5.3.13	sc_rm_set_master_attributes()	174
13.5.3.14	sc_rm_set_master_sid()	175
13.5.3.15	sc_rm_set_peripheral_permissions()	176
13.5.3.16	sc_rm_is_resource_owned()	176
13.5.3.17	sc_rm_get_resource_owner()	177
13.5.3.18	sc_rm_is_resource_master()	177
13.5.3.19	sc_rm_is_resource_peripheral()	178
13.5.3.20	sc_rm_get_resource_info()	178
13.5.3.21	sc_rm_memreg_alloc()	179
13.5.3.22	sc_rm_memreg_split()	180
13.5.3.23	sc_rm_memreg_frag()	180
13.5.3.24	sc_rm_memreg_free()	181
13.5.3.25	sc_rm_find_memreg()	182
13.5.3.26	sc_rm_assign_memreg()	182
13.5.3.27	sc_rm_set_memreg_permissions()	183
13.5.3.28	sc_rm_set_memreg_ide()	184
13.5.3.29	sc_rm_is_memreg_owned()	185
13.5.3.30	sc_rm_get_memreg_info()	185
13.5.3.31	sc_rm_assign_pad()	186
13.5.3.32	sc_rm_set_pad_movable()	186
13.5.3.33	sc_rm_is_pad_owned()	187
13.5.3.34	sc_rm_dump()	187
13.6	SECO: Security Service	188
13.6.1	Detailed Description	190

13.6.2 Function Documentation	192
13.6.2.1 sc_seco_image_load()	192
13.6.2.2 sc_seco_authenticate()	193
13.6.2.3 sc_seco_enh_authenticate()	194
13.6.2.4 sc_seco_forward_lifecycle()	195
13.6.2.5 sc_seco_return_lifecycle()	196
13.6.2.6 sc_seco_commit()	196
13.6.2.7 sc_seco_attest_mode()	197
13.6.2.8 sc_seco_attest()	198
13.6.2.9 sc_seco_get_attest_pkey()	198
13.6.2.10 sc_seco_get_attest_sign()	199
13.6.2.11 sc_seco_attest_verify()	200
13.6.2.12 sc_seco_gen_key_blob()	201
13.6.2.13 sc_seco_load_key()	202
13.6.2.14 sc_seco_get_mp_key()	202
13.6.2.15 sc_seco_update_mpmr()	203
13.6.2.16 sc_seco_get_mp_sign()	204
13.6.2.17 sc_seco_build_info()	205
13.6.2.18 sc_seco_chip_info()	205
13.6.2.19 sc_seco_enable_debug()	206
13.6.2.20 sc_seco_get_event()	206
13.6.2.21 sc_seco_fuse_write()	207
13.6.2.22 sc_seco_patch()	207
13.6.2.23 sc_seco_set_mono_counter_partition()	208
13.6.2.24 sc_seco_set_fips_mode()	209
13.6.2.25 sc_seco_start_rng()	210
13.6.2.26 sc_seco_sab_msg()	210
13.6.2.27 sc_seco_secvio_enable()	211
13.6.2.28 sc_seco_secvio_config()	212
13.6.2.29 sc_seco_secvio_dgo_config()	213
13.7 TIMER: Timer Service	214
13.7.1 Detailed Description	215
13.7.2 Function Documentation	215
13.7.2.1 sc_timer_set_wdog_timeout()	215
13.7.2.2 sc_timer_set_wdog_pre_timeout()	216
13.7.2.3 sc_timer_set_wdog_window()	216
13.7.2.4 sc_timer_start_wdog()	217
13.7.2.5 sc_timer_stop_wdog()	217
13.7.2.6 sc_timer_ping_wdog()	218

13.7.2.7 sc_timer_get_wdog_status()	218
13.7.2.8 sc_timer_pt_get_wdog_status()	219
13.7.2.9 sc_timer_set_wdog_action()	219
13.7.2.10 sc_timer_set_rtc_time()	220
13.7.2.11 sc_timer_get_rtc_time()	221
13.7.2.12 sc_timer_get_rtc_sec1970()	221
13.7.2.13 sc_timer_set_rtc_alarm()	222
13.7.2.14 sc_timer_set_rtc_periodic_alarm()	222
13.7.2.15 sc_timer_cancel_rtc_alarm()	223
13.7.2.16 sc_timer_set_rtc_calb()	223
13.7.2.17 sc_timer_set_sysctr_alarm()	224
13.7.2.18 sc_timer_set_sysctr_periodic_alarm()	224
13.7.2.19 sc_timer_cancel_sysctr_alarm()	225
14 File Documentation	227
14.1 platform/config/mx8qx/pads.h File Reference	227
14.1.1 Detailed Description	233
14.2 platform/main/ipc.h File Reference	233
14.2.1 Detailed Description	234
14.2.2 Function Documentation	234
14.2.2.1 sc_ipc_open()	234
14.2.2.2 sc_ipc_close()	234
14.2.2.3 sc_ipc_read()	234
14.2.2.4 sc_ipc_write()	235
14.3 platform/main/types.h File Reference	235
14.3.1 Detailed Description	252
14.3.2 Macro Definition Documentation	252
14.3.2.1 SC_R_NONE	252
14.3.2.2 SC_P_ALL	252
14.3.3 Typedef Documentation	252
14.3.3.1 sc_rsrc_t	252
14.3.3.2 sc_pad_t	253
14.4 platform/svc/irq/api.h File Reference	253
14.4.1 Detailed Description	255
14.5 platform/svc/misc/api.h File Reference	255
14.5.1 Detailed Description	258
14.6 platform/svc/pad/api.h File Reference	258
14.6.1 Detailed Description	261
14.7 platform/svc/pm/api.h File Reference	261

14.7.1 Detailed Description	265
14.8 platform/svc/rm/api.h File Reference	266
14.8.1 Detailed Description	269
14.9 platform/svc/seco/api.h File Reference	269
14.9.1 Detailed Description	271
14.10 platform/svc/timer/api.h File Reference	271
14.10.1 Detailed Description	273
Index	275

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](#)
- [Security](#)
- [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
- Some features of SNVS such as the RTC, button, and LPGPR1
- Any other resources reserved by the port of the SCFW to the board

Note also the SECO uses some resources like CAAM JR0-1, SNVS LPGPR0, etc.

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. More detail can be found in the [Power Management](#) section. 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.

Partitions own resources, pads, and memory regions. This includes owning MU resources to communicate with the SCFW. API calls to the SCFW lookup the owner of the MU the call comes in on, and that is treated as the calling partition. Different API calls then enforce operations based on the relationship of the calling partition to the partition being acted on. This association does not directly determine who can access the programming model of a resource IP. This is solely determined by the access rights defined (per partition) for the resource. Note partitions do not have owners (they are the owners), but they do have parent/child relationships. The creator of a partition is the parent unless that parent calls the API to change the parent. If no parent is desired, then the parent should be set to 0 (the SCFW itself). Being the parent provides some rights with respect to API calling (but not peripheral access).

More detail can be found in the [Resource Management](#) section. 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 Security Service

The SC firmware provides access to many security functions including:

- Image Authentication
- Generating, exporting, and loading key blobs
- Fuse programming
- Lifecycle management
- Attestation

The Security (SECO) API is documented [here](#).

See the SECO API Reference Guide for more info.

1.3.7 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.
- **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) 2020 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.

3.4 CPU Start Address

The execution address passed in the boot container is used by the SCFW to start the CPU. This is done by calling [sc_pm_boot\(\)](#) or [sc_pm_cpu_start\(\)](#), depending on the partition info specified in the container. This address is restricted by the hardware implementation.

3.4.1 Cortex-A Start Address

These cores are restricted to the following. Note the allowed address is CPU dependent.

Address	CPU
0x00000000	Any
0x00000040	1
0x00000080	2
0x000000C0	3
0x00000100	0
0x00000140	1
0x00000180	2
0x000001C0	3
0x00010000	1
0x00020000	2
0x00030000	3
0x80000000	Any
0x80000040	1
0x80000080	2
0x800000C0	3
0x80000100	0
0x80000140	1
0x80000180	2
0x800001C0	3
0x80010000	1
0x80020000	2
0x80030000	3

Some devices alias OCRAM to ROM. On these, if an OCRAM address is specified the SCFW will modify it to a ROM address. This will allow the code to be fetched from the OCRAM. Be warned the PC will be in the ROM address space.

3.4.2 Cortex-M Start Address

The hardware has no mechanism to set the boot address. These cores will always start at the beginning of TCM. The SCFW will take any other address specified and write a jump instruction to the beginning of TCM.

3.5 Cortex-M4 DDR Aliasing

Devices without a SMMU include a basic translation block that can be leveraged to alias DDR memory onto the Cortex-M4 code bus. This aliasing allows performance improvements when Cortex-M4 code is located in DDR memory. The Cortex-M4 image can active aliasing during execution by updating the MCM Process ID register (MCM_PID). The DDR region will be aliased to the FlexSPI0 memory-mapped (XIP) space.

The following mappings are supported:

MCM_PID	DDR Region	Size	Alias
0x7E	0x88000000 - 0x8FFFFFFF	128MB	0x08000000 - 0x0FFFFFFF
0x7F	0x90000000 - 0x97FFFFFF	128MB	0x10000000 - 0x17FFFFFF
Other	N/A	N/A	No Alias

Aliasing must be activated while executing from a non-aliased region. The recommended flow to boot the Cortex-M4 into an aliased DDR region is as follows:

- Link the image for the aliased region
- Reset vector must point to unaliased DDR start address
- Specify the load address to be unaliased DDR region in the boot container
- After Cortex-M4 execution begins from unaliased DDR, jump to the aliased region

Notes:

- Enabling aliasing via the MCM_PID will result in loss of access to the corresponding FlexSPI address space.
- Use caution when using the Cortex-M4 code and system bus caches while aliasing is active to avoid coherency issues.

Chapter 4

Resource Management

SCFW is responsible for managing ownership and access permissions to system resources. The resource management functions of the SCFW are used to divide up the System on a Chip (SoC) resources and to control master transaction attributes and peripheral access permissions. The features supported by the SCFW are:

- 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.
- Allows owners to configure access permissions to resources.
- Provides hardware enforced isolation.

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

4.1 Partitions

One of the primary concepts of resource management in the SCFW is resource partitions (aka partitions). These are used to define a 'logical SoC' made up of resources, pads, and memory regions.

Partitions have the following characteristics. All partitions:

- can be created and destroyed, at boot and dynamically during run-time,
- have a hierarchical relationship, every partition has a parent,
- own resources (master and peripheral), pads, and memory regions,
- are restricted, by default, to only accessing the resources they own,
- can be booted, rebooted, and powered down,
- have a power state; can transition to other power states,
- have a watchdog timer, RTC alarm, and system counter alarm.

Partitions are created by calling the SCFW to allocate them. Resources, pads, memory regions, etc. are then assigned or moved to the new partition thus defining the 'logical SoC'. The SCFW uses an assignment scheme for resource management rather than an allocation scheme. This is similar to virtualization and VMs.

At boot all resources belong to one partition. The SCFW creates some partitions in the board porting layer and then others are created dynamically by the running CPUs. Partitions should be created in the SCFW for any CPU that will be started by the SCFW due to parameters passed from the ROM and defined in the boot image containers.

Partitions can be created in several ways. They can be created by default by the SCFW using parameters passed from the ROM (determined by parameters in the boot container), they can be created in the `board_system_config()` function of the board porting layer of the SCFW, or be created dynamically by calling `sc_rm_partition_alloc()`.

Partitions have several attributes that are defined when they are created:

- **Secure** - by default, master would generate secure transactions and resources would require secure access. Only a secure partition can create another secure partition. Used primarily for TrustZone.
- **Isolated** - use XRDC2 to isolate. Should be true for all partitions except when a secure partition creates a non-secure partitions running on the same CPU as a secure partition. This parameter just results in the partition having the same DID as the parent.
- **Restricted** - true if the partition cannot create partitions.
- **Grant** - true if all resources in this partition should always remain accessible to the parent. Only used when creating a partition with no CPUs. Useful to just isolate the access of a bus master like a DMA channel to a subset of memory.
- **Coherent** - true if this partition will contain both AP clusters running in an SMP configuration.

The boot partition parameters come from the container. See the [Boot Flags](#) section.

In addition, partitions can be made confidential using the `sc_rm_set_confidential()` function. A confidential partition cannot not have any resource or memory assigned to it anymore. Useful for some security related use-cases.

The parent/child relationship directly affects partition reboot. When a partition is rebooted, its child partitions are automatically deleted and all resources returned to the parent. This allows the parent to rerun and again recreate the child partition like it did when it ran the first time. For more information on resets and partition reboot, see the Reset section of the Porting Guide.

4.2 Resources

Resources represent the IP blocks in the SoC. They can be masters (i.e. generate bus transactions), peripherals (i.e. have a programming model), or both. Resources always have an owning (only one) partition. Access control of resources consists of two primary mechanisms:

- **API Access** - the SCFW API functions use info like the calling partition and resource parameter to decide if actions can be take on a resource. The ownership of the resource and hierarchical relationship of the owning partition and the calling partition are used to decide if operations are authorized. These access rights vary by function and are described in the function documentation.

- **Hardware Protection** - the SCFW programs the underlying [protection hardware](#) (XRDC2 in the case of i.MX8) to control which masters can access which peripherals. Access permissions are maintained for each resource and can be set by the owner for each accessing partition. By default, only masters in a partition can only access peripherals in the same partition.

Resources can be assigned or moved by the owner to another partition. Master resources can have security, privilege, SMMU bypass, and streamID (SID) set. Peripheral resources can have access permissions set. The security state of masters can only be set if the partition owning the master is secure. Master resources generate a domain ID (DID) on the bus identifying what partition they belong to. All master resources in a partition generate the same ID so they all have the same access rights (CPUs and all other bus masters). For example, a DC assigned to a partition can only access memory accessible to that partition. The partition needs to either own the memory or the memory needs to have access rights granted to that partition using [sc_rm_set_memreg_permissions\(\)](#). If this needs to be a subset of memory then a new memory region should be created using one of the split or frag functions.

Note that not all resources have independent access controls. This is a function of the hardware design. Some peripherals share access control programming. This is often the case for display and camera interfaces. The entire interface (interface IP, PWM, I2C, LPCG, etc.) are in a single 64K page and have a single access control slot.

While Cortex-A cores part of a cluster can be assigned to different partitions, this is not guaranteed to always identify bus transactions as belonging to a core. Due to variations in microarchitecture of different Cortex-A cores, they have varying abilities to identify traffic as belonging to a specific core. This ability depends on transaction type (strongly ordered, device, exclusive, normal, etc.) as well as L1/L2 cache usage. Because cores can define these transaction properties themselves, isolation between CPUs in a cluster can never be considered secure. Because of this, i.MX8 processors do not have a product requirement to support this capability and the ability to isolate cores is not tested. CPUs in a cluster should always be in the same partition as the cluster.

4.3 Pads

Pads represent the individual pads of the SoC. They are owned by partitions. They have no direct access, so access control is similar to the SCFW API access control of resources.

4.4 Memory Regions

Memory regions represent blocks of memory with a start and end address. Once defined, they have ownership and access controls similar to resources. They support both kinds of access protections. The memory region owner can:

- assign/move to another partition
- can create a new memory region within the bounds of an existing owned region
- can split a region
- can configure access permissions

The SCFW implementation supports a maximum of 64 memory regions. Beyond this, the HW implementation further restricts the number of regions and this restriction varies depending on what HW checker is used for each memory and how many regions that specific checker supports. For example, most i.MX8 SoC have a limit of 16 regions for the DDR address range. Of these 16, one is used by the SCFW to define a I/O space pass through so the effective limit for SCFW API use is 15 unique DDR regions. Also, the SCFW will optimize usage of the HW regions. For example, if two regions are coincident, it will only use one HW region to enforce the access policy. These totals are for the entire system.

4.5 SCFW API

CPUs in partitions make SCFW API calls via a [remote procedure call \(RPC\) mechanism](#). The RPC mechanism serializes the call and sends it over an inter-processor communication (IPC) mechanism implemented via message units (MU). Message units are resources and since all resources are owned by a partition, the SCFW can identify the owner and hence determine the calling partition. The calling partition is often used to determine if an operation is allowed or not.

The SCFW has the following services that operate on partitions and resources:

- **Resource Management (RM)** - used to manage partitions, resources, pads, memory regions
- **Power Management (PM)** - used to boot, reboot, and change power state of partitions and resources
- **Timer** - used to configure and use partitions-specific watchdogs and alarms
- **Interrupt (IRQ)** - used to manage interrupts sent to partitions via their MUs

The other services (pad, miscellaneous, and security) do not operate on partitions or resources. The Resource Management (RM) API is documented [here](#).

Note there are eight MUs (five in LSIO, one in each M4, and one in the SCU itself) that are used to communicate to the SCFW). This is because one end of each of these is accessible only via a private bus used by the SCU.

- SC_R_MU_0A-4A (A side used by CPUs, B-side used by SCU)
- SC_R_M4_0_MU_1A (A side used by CPU, B-side used by SCU)
- SC_R_M4_1_MU_1A (A side used by CPU, B-side used by SCU)
- SC_R_SC_MU_1A in the SCU (both sides used by SCU for loopback testing)

The SCFW does not dictate which CPU uses each of these. Access is controlled by the standard SCFW RM partitioning. SCFW has to power at least one up for each CPU and these are specified in the boot container. The default in mkimage is the M4 MU for each M4 and MU_0A for the AP.

Beyond this, all usage is determined by the board.c port and the AP/M4 SW. Access to MUs is controlled by the SCFW RM partitioning which is configured in board.c and at run-time by the SW themselves. Typical usage is MU_0A is kept and used by the AP secure code and MU_1A is used by the AP non-secure code. Using AP clusters to run separate OSes or using a hypervisor would require additional usage of these MUs.

4.6 Hardware Resource/Memory Isolation

The i.MX8 SoC contains a mix of Cortex-A and Cortex-M CPUs which frequently operate in an asymmetric mode with different software environments executing on them (OSes, RTOSes, etc.). To keep these SW environments from unintentionally interfering with each other, i.MX8 contains HW mechanisms to enforce isolation. These mechanisms operate in a manner similar to ARM's TrustZone in that transactions from masters are annotated with user-side band information to indicate their domain and the access controls allow/disallow access to peripherals/memory based on this information.

The HW that does this is the XRDC2 (aka XRDC). The XRDC consists of a manager (per subsystem), MDAC, PAC, MSC, and MRC components. The MDAC is used to annotate the transactions and the PAC, MSC, and MRC are used to control access. See the figure below for the basic XRDC integration architecture. In i.MX8, the XRDC replaces the RDC and TrustZone components (CSU, TZASC, etc.) found in previous i.MX processors.

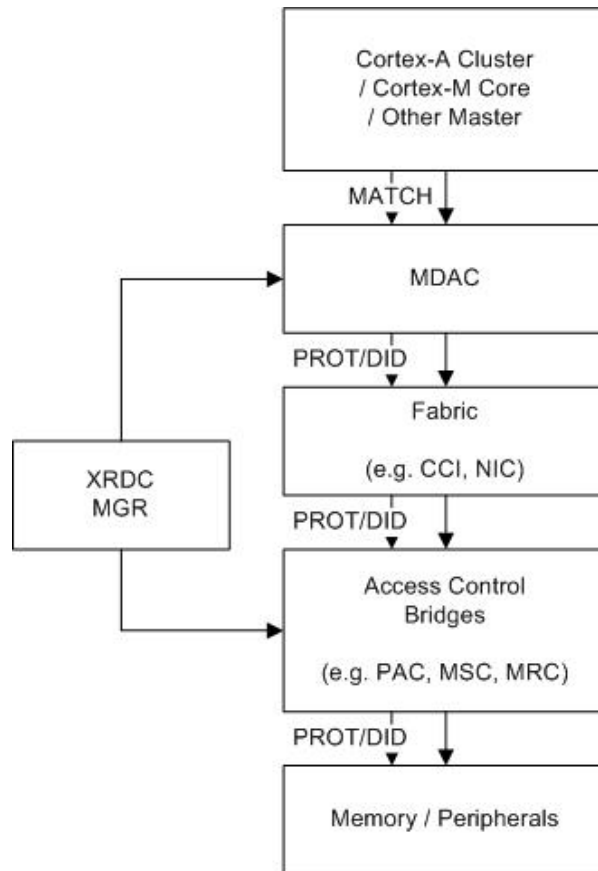


Figure 4.1 XRDC Interaction for Bus Transactions

There are five XRDC components:

- **Manager (MGR)** - provides the programming interface for the entire XRDC
- **Master Domain Assignment Controller (MDAC)** - identifies transaction sources/types and appends information such as domain ID (DID), streamID (SID), etc.
- **Peripheral Access Controller (PAC)** - accepts/rejects transactions based on transaction attributes and programmed permissions; supports up to 128 different peripherals (IPS or APB based); based on module enables and/or select signals so has no concept of addressing
- **Memory Space Controller (MSC)** - accepts/rejects transactions based on transaction attributes and programmed permissions; supports a single memory space; has no concept of addressing
- **Memory Region Controller (MRC)** - accepts/rejects transactions based on transaction attributes and programmed permissions; supports multiple memory spaces by allowing the overall space to be divided into regions (with programmable start/end addresses)

The SCFW configures the XRDC2 components based on partition, resource, and memory region configuration. For more information on the XRDC capabilities, refer to the XRDC2 section of the RM.

4.7 Example

The following shows the partition creation for an example system. Initially, the SCFW creates three partitions. The SCFW and SECO partitions are secure and isolated. The boot partition parameters depend on the boot [flags](#) from the boot image container. These would also normally be secure and isolated. The boot partition contains all the memory and almost all the resources. Only the minimal resources required for the SCFW and SECO to function are owned by those partitions.

Initial RM partition state:

- Partition 0: SCFW
- Partition 1: Boot partition
- Partition 2: SECO

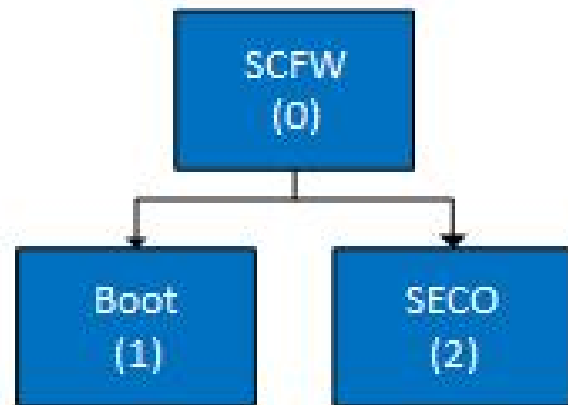


Figure 4.2 Initial Partition Configuration

Before starting the CPUs, the SCFW calls `board_system_config()`. This is where the partitions are created for CPUs started by the SCFW. These partitions usually have code loaded by the ROM and execution automatically started by the SCFW. Partitions can also be created for CPUs that aren't started immediately by the SCFW. Code can be loaded for these partitions by the ROM at boot by defining them as data images with `mkimage`. Code could also be loaded by the parent so long as it has access rights to the memory. The parent can then boot such a partition later by calling `sc_pm_boot()`. An M4 partition would normally be non-secure.

RM partition state:

- Partition 0: SCFW
- Partition 1: ATF
- Partition 2: SECO
- Partition 3: M4

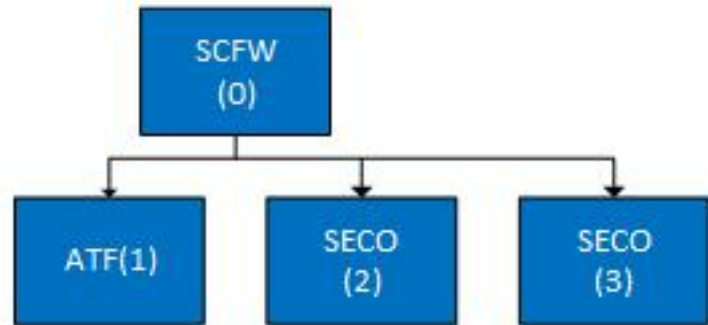


Figure 4.3 Board Partition Configuration

The boot partition can also be considered the ATF partition because that is the first thing run on the AP core. ATF creates a partition for Linux to use. This partition would typically be non-secure and not isolated. The Cortex-A CPUs, MU0A, the SC_R_SYSTEM resource, and a little bit of memory are kept by the ATF partition. All other resources are assigned to the Linux partition.

RM partition state:

- Partition 0: SCFW
- Partition 1: ATF
- Partition 2: SECO
- Partition 3: M4
- Partition 4: Linux

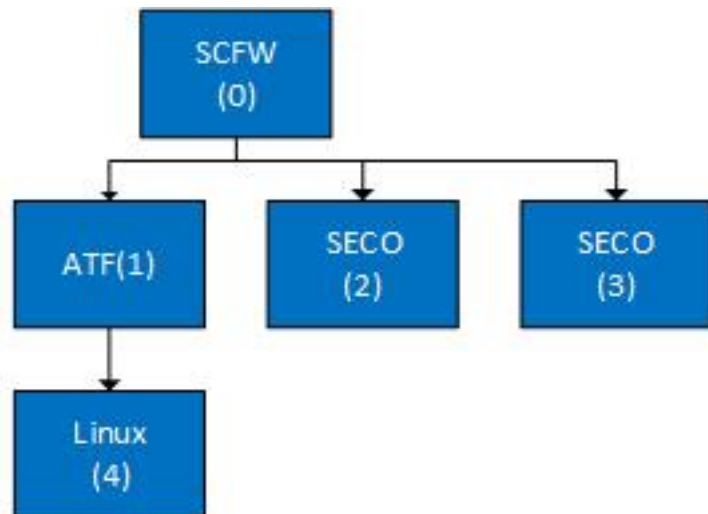


Figure 4.4 Final Partition Configuration

ATF starts Linux. Linux could dynamically create an additional partition for the other M4 to handle something like sensor fusion or a media processing engine.

Chapter 5

Power Management

SCFW is responsible for managing the power state of the SoC. This includes static and dynamic power management, clock management, and reset control. The features supported by the SCFW are:

- Changing the power state of the system, partitions, and resources.
- Configuring clocks and PLLs.
- Configuring and responding to wake-up sources.
- Reset control including booting/rebooting a partition, obtaining reset reasons, and starting/stopping of CPUs.

See the [Overview](#). The Power Management (PM) API is documented [here](#).

5.1 Wake from Low Power

5.1.1 Wake Event Sources

The SoC RM provides some background on the wakeup capability of the IRQSTEER and GIC modules. The wakeup outputs of the IRQSTEER and GIC are connected to IRQ lines of the SCU and fielded by SCFW to wake up the respective CPU for wake event processing. The [SCFW Interrupt Service](#) can also generate wakeup events that do not require the GIC or IRQSTEER to remain active. Each of these wake events sources are described in the sections below. Note the tradeoff between flexibility of the wakeup method and power consumption that must be considered during system design.

5.1.2 GIC Wake Events

The GIC module provides a facility to transition each redistributor interface into a sleep state with wakeup capability. AP software uses the GICR_WAKER register to quiesce the respective redistributor interface. Prior to executing WFI, AP software notifies SCFW of the request to enter a deeper low-power mode using [sc_pm_req_cpu_low_power_mode\(\)](#). Note that the GIC must be powered and clocked to generate a wake event. The following table shows the GIC wakeup options available:

sc_pm_req_cpu_low_power_↔ mode wake_src parameter	Wakeup Description	Notes
SC_PM_SRC_GIC	GIC wakeup will cause SCFW to wake the respective CPU	GIC resource must remain powered and clocked (power mode \geq LP). System power consumption will be higher to keep respective subsystem with GIC powered and clocked. K↔S1 power mode is inhibited with this wake method.
SC_PM_WAKE_SRC_IRQSTEE↔ R_GIC	Refer to IRQSTEER wake events in the next section	

5.1.3 IRQSTEER Wake Events

One of the IRQSTEER module instances (SC_R_IRQSTR_SCU2) is reserved for AP wake events. The IRQST↔EER does not require clocks to generate a wakeup event and therefore offers lower power consumption than GIC wakeup methods. Prior to executing WFI, AP software notifies SCFW of the request to enter a deeper low-power mode using [sc_pm_req_cpu_low_power_mode\(\)](#). Note that the IRQSTEER must be powered to generate a wake event. The following table shows the IRQSTEER wakeup options available:

sc_pm_req_cpu_low_power_↔ mode wake_src parameter	Wakeup Description	Notes
SC_PM_WAKE_SRC_IRQSTEE↔ R_GIC	2-stage wakeup with first stage being IRQSTEER and second stage being GIC	GIC and IRQSTEER resources must remain powered (power mode \geq STBY). AP software must "replicate" the desired wake sources from the GIC to the IRQSTEER. System power consumption will be higher to keep respective subsystem with G↔IC and IRQSTEER powered. K↔S1 power mode is inhibited with this wake method.
SC_PM_WAKE_SRC_IRQSTEER	IRQSTEER wakeup will cause SC↔FW to wake the "primary" CPU	IRQSTEER resource must remain powered (power mode \geq STBY). System power consumption will be higher to keep respective subsystem with IRQSTEER powered. K↔S1 power mode is inhibited with this wake method. Primary CPU designated using sc_pm_set_cpu↔_resume .

5.1.4 SCU Wake Events

SCU wake events allow the entire system to be placed into retention/powered down. These wake sources are enabled using the SCFW IRQ service API ([sc_irq_enable](#)). Prior to executing WFI, AP software notifies SCFW of the request to enter a deeper low-power mode using [sc_pm_req_cpu_low_power_mode\(\)](#). The following table shows the SCU wakeup options available:

sc_pm_req_cpu_low_power_↔ mode wake_src parameter	Wakeup Description	Notes
SC_PM_WAKE_SRC_SCU	SCU wake event will wake the "primary" CPU	GIC and IRQSTEER resources can be powered down. KS1 power mode is possible with this wake method.

A summary of SCU wake events supported by SCFW:

- RTC
- ON_OFF button
- Pad event
- System Counter

5.1.5 Pad Wake Events

The pad modules provide a programmable wakeup capability. Software must enable and configure the respective pad to generate a wakeup using the SCFW IRQ Service API. The following is an example of the calls required to enable and service a pad wake events for the UART RX signal (note that SCFW API return error checking removed for simplicity).

```
/* Register for PAD wakeup */
sc_irq_enable(ipc, MU_RSRC, SC_IRQ_GROUP_WAKE, SC_IRQ_PAD, SC_TRUE);
/* Enable UART_RX pad wakeup */
sc_pad_set_wakeup(ipc, UART_RX_PAD, SC_PAD_WAKEUP_LOW_LVL);
/* Enter low-power mode and wait for wakeup... */
/* Query SCU wakeup event status */
uint32_t status;
sc_irq_status(ipc, MU_RSRC, SC_IRQ_GROUP_WAKE, &status);
/* Check for pad wakeup */
if (status & SC_IRQ_PAD)
{
    /* Check for UART_RX pad wakeup */
    /* Note: SCFW updates pending pad wakeup config to SC_PAD_WAKEUP_OFF */
    sc_pad_wakeup_t uart_wakeup;
    sc_pad_get_wakeup(ipc, UART_RX_PAD, &uart_wakeup);
    if (uart_wakeup == SC_PAD_WAKEUP_OFF)
    {
        /* UART_RX pad generated wake event */
    }
    else
    {
        /* Else some other pad caused the wake event */
        /* Disable the UART_RX pad wakeup */
        sc_pad_set_wakeup(ipc, UART_RX_PAD, SC_PAD_WAKEUP_OFF);
    }
}
```

5.2 System Power Off

The system can be powered off by calling `sc_pm_set_sys_power_mode()` with mode = SC_PM_PW_MODE_OFF. This calls `board_power()` in `board.c` to allow for port specific control of the PMIC. Note only a caller with access to the SC_R_SYSTEM resource can call this function.

5.3 Reset Control

For more info on resets, see the Reset section. This includes booting and rebooting partitions, rebooting the system, starting/stopping CPUs, and watchdog usage.

Chapter 6

Resource List

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

The *_PIDx resources represent multiple contexts that the master can act as (aka process IDs). The master functions normally as PID0 but can switch to other PIDs if it needs to access the resources of another SW system. This is primarily used to support virtualization. Each master PID resource can be assigned a different SID and as a result a different translation context in the SMMU. If not using virtualization, all PIDs for a resource should be assigned to the partition owning the master IP.

Note the resource define values (index) are listed here but the define should be used. These are defined in [types.h](#). Functions like [sc_rm_set_resource_movable\(\)](#) take a resource range. This range is numerical by index and not alphabetical by define name. Range should be lowest to highest index.

See also

[RM: Resource Management Service](#)

[PM: Power Management Service](#)

Resource	Index	Subsystem	Instance	Description
SC_R_A35	507	A35	0	Cluster
SC_R_A35_0	508	A35	0	CPU 0
SC_R_A35_1	509	A35	0	CPU 1
SC_R_A35_2	510	A35	0	CPU 2
SC_R_A35_3	511	A35	0	CPU 3
SC_R_ADC_0	101	ADMA	0	ADC 0
SC_R_AMIX	458	ADMA	0	Audio mixer
SC_R_ASRC_0	414	ADMA	0	ASRC 0
SC_R_ASRC_1	454	ADMA	0	ASRC 1
SC_R_ATTESTATION	545	SC	0	Attestation
SC_R_AUDIO_CLK_0	493	ADMA	0	Audio clock 0
SC_R_AUDIO_CLK_1	494	ADMA	0	Audio clock 1
SC_R_AUDIO_PLL_0	325	ADMA	0	Audio PLL 0
SC_R_AUDIO_PLL_1	492	ADMA	0	Audio PLL 1
SC_R_BOARD_R0	524	BOARD	0	Misc. board component 0

Resource	Index	Subsystem	Instance	Description
SC_R_BOARD_R1	525	BOARD	0	Misc. board component 1
SC_R_BOARD_R2	526	BOARD	0	Misc. board component 2
SC_R_BOARD_R3	527	BOARD	0	Misc. board component 3
SC_R_BOARD_R4	528	BOARD	0	Misc. board component 4
SC_R_BOARD_R5	529	BOARD	0	Misc. board component 5
SC_R_BOARD_R6	530	BOARD	0	Misc. board component 6
SC_R_BOARD_R7	531	BOARD	0	Misc. board component 7
SC_R_CAAM_JR1	500	SC	0	CAAM job ring 1
SC_R_CAAM_JR1_OUT	514	SC	0	CAAM job ring 1 out
SC_R_CAAM_JR2	501	SC	0	CAAM job ring 2
SC_R_CAAM_JR2_OUT	515	SC	0	CAAM job ring 2 out
SC_R_CAAM_JR3	502	SC	0	CAAM job ring 3
SC_R_CAAM_JR3_OUT	516	SC	0	CAAM job ring 3 out
SC_R_CAN_0	105	ADMA	0	CAN 0
SC_R_CAN_1	106	ADMA	0	CAN 1
SC_R_CAN_2	107	ADMA	0	CAN 2
SC_R_CSI_0	401	CSI	0	CSI
SC_R_CSI_0_I2C_0	403	CSI	0	I2C
SC_R_CSI_0_PWM_0	402	CSI	0	PWM
SC_R_DB	11	DB	0	DB
SC_R_DC_0	32	DC	0	DC
SC_R_DC_0_BLIT0	19	DC	0	BLIT0
SC_R_DC_0_BLIT1	20	DC	0	BLIT1
SC_R_DC_0_BLIT2	21	DC	0	BLIT2
SC_R_DC_0_BLIT_OUT	22	DC	0	BLIT OUT
SC_R_DC_0_FRAC0	30	DC	0	FRAC0
SC_R_DC_0_PLL_0	34	DC	0	PLL 0
SC_R_DC_0_PLL_1	35	DC	0	PLL 1
SC_R_DC_0_VIDEO0	28	DC	0	VIDEO0
SC_R_DC_0_VIDEO1	29	DC	0	VIDEO1
SC_R_DC_0_WARP	25	DC	0	WARP
SC_R_DEBUG	354	SC	0	Debug
SC_R_DMA_0_CH0	64	ADMA	0	DMA 0 channel 0 (audio)
SC_R_DMA_0_CH1	65	ADMA	0	DMA 0 channel 1 (audio)
SC_R_DMA_0_CH10	74	ADMA	0	DMA 0 channel 10 (audio)
SC_R_DMA_0_CH11	75	ADMA	0	DMA 0 channel 11 (audio)
SC_R_DMA_0_CH12	76	ADMA	0	DMA 0 channel 12 (audio)
SC_R_DMA_0_CH13	77	ADMA	0	DMA 0 channel 13 (audio)
SC_R_DMA_0_CH14	78	ADMA	0	DMA 0 channel 14 (audio)
SC_R_DMA_0_CH15	79	ADMA	0	DMA 0 channel 15 (audio)
SC_R_DMA_0_CH16	80	ADMA	0	DMA 0 channel 16 (audio)
SC_R_DMA_0_CH17	81	ADMA	0	DMA 0 channel 17 (audio)
SC_R_DMA_0_CH18	82	ADMA	0	DMA 0 channel 18 (audio)
SC_R_DMA_0_CH19	83	ADMA	0	DMA 0 channel 19 (audio)
SC_R_DMA_0_CH2	66	ADMA	0	DMA 0 channel 2 (audio)
SC_R_DMA_0_CH20	84	ADMA	0	DMA 0 channel 20 (audio)

Resource	Index	Subsystem	Instance	Description
SC_R_DMA_0_CH21	85	ADMA	0	DMA 0 channel 21 (audio)
SC_R_DMA_0_CH22	86	ADMA	0	DMA 0 channel 22 (audio)
SC_R_DMA_0_CH23	87	ADMA	0	DMA 0 channel 23 (audio)
SC_R_DMA_0_CH24	88	ADMA	0	DMA 0 channel 24 (audio)
SC_R_DMA_0_CH25	89	ADMA	0	DMA 0 channel 25 (audio)
SC_R_DMA_0_CH26	90	ADMA	0	DMA 0 channel 26 (audio)
SC_R_DMA_0_CH27	91	ADMA	0	DMA 0 channel 27 (audio)
SC_R_DMA_0_CH28	92	ADMA	0	DMA 0 channel 28 (audio)
SC_R_DMA_0_CH29	93	ADMA	0	DMA 0 channel 29 (audio)
SC_R_DMA_0_CH3	67	ADMA	0	DMA 0 channel 3 (audio)
SC_R_DMA_0_CH30	94	ADMA	0	DMA 0 channel 30 (audio)
SC_R_DMA_0_CH31	95	ADMA	0	DMA 0 channel 31 (audio)
SC_R_DMA_0_CH4	68	ADMA	0	DMA 0 channel 4 (audio)
SC_R_DMA_0_CH5	69	ADMA	0	DMA 0 channel 5 (audio)
SC_R_DMA_0_CH6	70	ADMA	0	DMA 0 channel 6 (audio)
SC_R_DMA_0_CH7	71	ADMA	0	DMA 0 channel 7 (audio)
SC_R_DMA_0_CH8	72	ADMA	0	DMA 0 channel 8 (audio)
SC_R_DMA_0_CH9	73	ADMA	0	DMA 0 channel 9 (audio)
SC_R_DMA_1_CH0	108	ADMA	0	DMA 1 channel 0 (audio)
SC_R_DMA_1_CH1	109	ADMA	0	DMA 1 channel 1 (audio)
SC_R_DMA_1_CH10	118	ADMA	0	DMA 1 channel 10 (audio)
SC_R_DMA_1_CH11	119	ADMA	0	DMA 1 channel 11 (audio)
SC_R_DMA_1_CH12	120	ADMA	0	DMA 1 channel 12 (audio)
SC_R_DMA_1_CH13	121	ADMA	0	DMA 1 channel 13 (audio)
SC_R_DMA_1_CH14	122	ADMA	0	DMA 1 channel 14 (audio)
SC_R_DMA_1_CH15	123	ADMA	0	DMA 1 channel 15 (audio)
SC_R_DMA_1_CH2	110	ADMA	0	DMA 1 channel 2 (audio)
SC_R_DMA_1_CH3	111	ADMA	0	DMA 1 channel 3 (audio)
SC_R_DMA_1_CH4	112	ADMA	0	DMA 1 channel 4 (audio)
SC_R_DMA_1_CH5	113	ADMA	0	DMA 1 channel 5 (audio)
SC_R_DMA_1_CH6	114	ADMA	0	DMA 1 channel 6 (audio)
SC_R_DMA_1_CH7	115	ADMA	0	DMA 1 channel 7 (audio)
SC_R_DMA_1_CH8	116	ADMA	0	DMA 1 channel 8 (audio)
SC_R_DMA_1_CH9	117	ADMA	0	DMA 1 channel 9 (audio)
SC_R_DMA_2_CH0	254	ADMA	0	DMA 2 channel 0
SC_R_DMA_2_CH1	255	ADMA	0	DMA 2 channel 1
SC_R_DMA_2_CH10	432	ADMA	0	DMA 2 channel 10
SC_R_DMA_2_CH11	433	ADMA	0	DMA 2 channel 11
SC_R_DMA_2_CH12	434	ADMA	0	DMA 2 channel 12
SC_R_DMA_2_CH13	435	ADMA	0	DMA 2 channel 13
SC_R_DMA_2_CH14	436	ADMA	0	DMA 2 channel 14
SC_R_DMA_2_CH15	437	ADMA	0	DMA 2 channel 15
SC_R_DMA_2_CH16	438	ADMA	0	DMA 2 channel 16
SC_R_DMA_2_CH17	439	ADMA	0	DMA 2 channel 17
SC_R_DMA_2_CH18	440	ADMA	0	DMA 2 channel 18

Resource	Index	Subsystem	Instance	Description
SC_R_DMA_2_CH19	441	ADMA	0	DMA 2 channel 19
SC_R_DMA_2_CH2	256	ADMA	0	DMA 2 channel 2
SC_R_DMA_2_CH20	442	ADMA	0	DMA 2 channel 20
SC_R_DMA_2_CH21	443	ADMA	0	DMA 2 channel 21
SC_R_DMA_2_CH22	444	ADMA	0	DMA 2 channel 22
SC_R_DMA_2_CH23	445	ADMA	0	DMA 2 channel 23
SC_R_DMA_2_CH24	446	ADMA	0	DMA 2 channel 24
SC_R_DMA_2_CH25	447	ADMA	0	DMA 2 channel 25
SC_R_DMA_2_CH26	448	ADMA	0	DMA 2 channel 26
SC_R_DMA_2_CH27	449	ADMA	0	DMA 2 channel 27
SC_R_DMA_2_CH28	450	ADMA	0	DMA 2 channel 28
SC_R_DMA_2_CH29	451	ADMA	0	DMA 2 channel 29
SC_R_DMA_2_CH3	257	ADMA	0	DMA 2 channel 3
SC_R_DMA_2_CH30	452	ADMA	0	DMA 2 channel 30
SC_R_DMA_2_CH31	453	ADMA	0	DMA 2 channel 31
SC_R_DMA_2_CH4	258	ADMA	0	DMA 2 channel 4
SC_R_DMA_2_CH5	427	ADMA	0	DMA 2 channel 5
SC_R_DMA_2_CH6	428	ADMA	0	DMA 2 channel 6
SC_R_DMA_2_CH7	429	ADMA	0	DMA 2 channel 7
SC_R_DMA_2_CH8	430	ADMA	0	DMA 2 channel 8
SC_R_DMA_2_CH9	431	ADMA	0	DMA 2 channel 9
SC_R_DMA_3_CH0	460	ADMA	0	DMA 3 channel 0
SC_R_DMA_3_CH1	461	ADMA	0	DMA 3 channel 1
SC_R_DMA_3_CH10	470	ADMA	0	DMA 3 channel 10
SC_R_DMA_3_CH11	471	ADMA	0	DMA 3 channel 11
SC_R_DMA_3_CH12	472	ADMA	0	DMA 3 channel 12
SC_R_DMA_3_CH13	473	ADMA	0	DMA 3 channel 13
SC_R_DMA_3_CH14	474	ADMA	0	DMA 3 channel 14
SC_R_DMA_3_CH15	475	ADMA	0	DMA 3 channel 15
SC_R_DMA_3_CH2	462	ADMA	0	DMA 3 channel 2
SC_R_DMA_3_CH3	463	ADMA	0	DMA 3 channel 3
SC_R_DMA_3_CH4	464	ADMA	0	DMA 3 channel 4
SC_R_DMA_3_CH5	465	ADMA	0	DMA 3 channel 5
SC_R_DMA_3_CH6	466	ADMA	0	DMA 3 channel 6
SC_R_DMA_3_CH7	467	ADMA	0	DMA 3 channel 7
SC_R_DMA_3_CH8	468	ADMA	0	DMA 3 channel 8
SC_R_DMA_3_CH9	469	ADMA	0	DMA 3 channel 9
SC_R_DMA_4_CH0	372	CONN	0	DMA channel 0
SC_R_DMA_4_CH1	373	CONN	0	DMA channel 1
SC_R_DMA_4_CH2	374	CONN	0	DMA channel 2
SC_R_DMA_4_CH3	375	CONN	0	DMA channel 3
SC_R_DMA_4_CH4	376	CONN	0	DMA channel 4
SC_R_DRC_0	12	DRC	0	DDR controller/phy
SC_R_DSP	512	ADMA	0	DSP
SC_R_DSP_RAM	513	ADMA	0	DSP RAM
SC_R_DTCP	264	CONN	0	DTCP
SC_R_ELCDIF_PLL	323	ADMA	0	eLCDIF PLL

Resource	Index	Subsystem	Instance	Description
SC_R_ENET_0	251	CONN	0	ENET 1
SC_R_ENET_1	252	CONN	0	ENET 2
SC_R_ESAI_0	415	ADMA	0	ESAI 0
SC_R_FSPI_0	237	LSIO	0	Flexible Serial Peripheral Interface (FSPI) 0
SC_R_FSPI_1	238	LSIO	0	FSPI 1
SC_R_FTM_0	103	ADMA	0	FTM 0
SC_R_FTM_1	104	ADMA	0	FTM 1
SC_R_GIC	18	ADMA	0	GIC
SC_R_GPIO_0	199	LSIO	0	General Purpose I/O (GPIO) 0
SC_R_GPIO_1	200	LSIO	0	GPIO 1
SC_R_GPIO_2	201	LSIO	0	GPIO 2
SC_R_GPIO_3	202	LSIO	0	GPIO 3
SC_R_GPIO_4	203	LSIO	0	GPIO 4
SC_R_GPIO_5	204	LSIO	0	GPIO 5
SC_R_GPIO_6	205	LSIO	0	GPIO 6
SC_R_GPIO_7	206	LSIO	0	GPIO 7
SC_R_GPT_0	207	LSIO	0	General Purpose Timer (GPT) 0
SC_R_GPT_1	208	LSIO	0	GPT 1
SC_R_GPT_10	426	ADMA	0	GPT 5
SC_R_GPT_2	209	LSIO	0	GPT 2
SC_R_GPT_3	210	LSIO	0	GPT 3
SC_R_GPT_4	211	LSIO	0	GPT 4
SC_R_GPT_5	421	ADMA	0	GPT 0
SC_R_GPT_6	422	ADMA	0	GPT 1
SC_R_GPT_7	423	ADMA	0	GPT 2
SC_R_GPT_8	424	ADMA	0	GPT 3
SC_R_GPT_9	425	ADMA	0	GPT 4
SC_R_GPU_0_PID0	144	GPU	0	PID0
SC_R_GPU_0_PID1	145	GPU	0	PID1
SC_R_GPU_0_PID2	146	GPU	0	PID2
SC_R_GPU_0_PID3	147	GPU	0	PID3
SC_R_HSIO_GPIO	172	HSIO	0	GPIO/MISC
SC_R_I2C_0	96	ADMA	0	I2C 0
SC_R_I2C_1	97	ADMA	0	I2C 1
SC_R_I2C_2	98	ADMA	0	I2C 2
SC_R_I2C_3	99	ADMA	0	I2C 3
SC_R_IEE	239	LSIO	0	Inline Encryption Engine (IEE)
SC_R_IEE_R0	240	LSIO	0	IEE region 0
SC_R_IEE_R1	241	LSIO	0	IEE region 1
SC_R_IEE_R2	242	LSIO	0	IEE region 2
SC_R_IEE_R3	243	LSIO	0	IEE region 3
SC_R_IEE_R4	244	LSIO	0	IEE region 4
SC_R_IEE_R5	245	LSIO	0	IEE region 5
SC_R_IEE_R6	246	LSIO	0	IEE region 6
SC_R_IEE_R7	247	LSIO	0	IEE region 7
SC_R_IRQSTR_DSP	322	ADMA	0	IRQSTR DSP
SC_R_IRQSTR_M4_0	15	ADMA	0	IRQSTR M4 0

Resource	Index	Subsystem	Instance	Description
SC_R_IRQSTR_M4_1	16	ADMA	0	IRQSTR M4 1 (optional)
SC_R_IRQSTR_SCU2	321	ADMA	0	IRQSTR SCU2/WAKE
SC_R_ISI_CH0	377	IMG	0	ISI channel 0
SC_R_ISI_CH1	378	IMG	0	ISI channel 1
SC_R_ISI_CH2	379	IMG	0	ISI channel 2
SC_R_ISI_CH3	380	IMG	0	ISI channel 3
SC_R_ISI_CH4	381	IMG	0	ISI channel 4
SC_R_ISI_CH5	382	IMG	0	ISI channel 5
SC_R_ISI_CH6	383	IMG	0	ISI channel 6
SC_R_ISI_CH7	384	IMG	0	ISI channel 7
SC_R_KPP	212	LSIO	0	Keypad Port (KPP)
SC_R_LCD_0	187	ADMA	0	eLCDIF 0
SC_R_LCD_0_PWM_0	188	ADMA	0	PWM 0
SC_R_LVDS_0	266	MIPI	0	LVDS
SC_R_LVDS_1	270	MIPI	1	LVDS
SC_R_M4_0_I2C	288	M4	0	I2C
SC_R_M4_0_INTMUX	289	M4	0	INTMUX
SC_R_M4_0_MU_0A0	293	M4	0	MU 0A0
SC_R_M4_0_MU_0A1	294	M4	0	MU 0A1
SC_R_M4_0_MU_0A2	295	M4	0	MU 0A2
SC_R_M4_0_MU_0A3	296	M4	0	MU 0A3
SC_R_M4_0_MU_0B	292	M4	0	MU 0B
SC_R_M4_0_MU_1A	297	M4	0	MU 1A
SC_R_M4_0_PID0	278	M4	0	PID0
SC_R_M4_0_PID1	279	M4	0	PID1
SC_R_M4_0_PID2	280	M4	0	PID2
SC_R_M4_0_PID3	281	M4	0	PID3
SC_R_M4_0_PID4	282	M4	0	PID4
SC_R_M4_0_PIT	286	M4	0	PIT
SC_R_M4_0_RGPIIO	283	M4	0	RGPIIO
SC_R_M4_0_SEMA42	284	M4	0	SEMA42
SC_R_M4_0_TPM	285	M4	0	TPM
SC_R_M4_0_UART	287	M4	0	UART
SC_R_MATCH_0	154	HSIO	0	Match 0
SC_R_MATCH_1	155	HSIO	0	Match 1
SC_R_MATCH_10	164	HSIO	0	Match 10
SC_R_MATCH_11	165	HSIO	0	Match 11
SC_R_MATCH_12	166	HSIO	0	Match 12
SC_R_MATCH_13	167	HSIO	0	Match 13
SC_R_MATCH_14	168	HSIO	0	Match 14
SC_R_MATCH_15	173	HSIO	0	Match 15
SC_R_MATCH_16	174	HSIO	0	Match 16
SC_R_MATCH_17	175	HSIO	0	Match 17
SC_R_MATCH_18	176	HSIO	0	Match 18
SC_R_MATCH_19	177	HSIO	0	Match 19
SC_R_MATCH_2	156	HSIO	0	Match 2
SC_R_MATCH_20	178	HSIO	0	Match 20

Resource	Index	Subsystem	Instance	Description
SC_R_MATCH_21	179	HSIO	0	Match 21
SC_R_MATCH_22	180	HSIO	0	Match 22
SC_R_MATCH_23	181	HSIO	0	Match 23
SC_R_MATCH_24	182	HSIO	0	Match 24
SC_R_MATCH_25	183	HSIO	0	Match 25
SC_R_MATCH_26	184	HSIO	0	Match 26
SC_R_MATCH_27	185	HSIO	0	Match 27
SC_R_MATCH_28	186	HSIO	0	Match 28
SC_R_MATCH_3	157	HSIO	0	Match 3
SC_R_MATCH_4	158	HSIO	0	Match 4
SC_R_MATCH_5	159	HSIO	0	Match 5
SC_R_MATCH_6	160	HSIO	0	Match 6
SC_R_MATCH_7	161	HSIO	0	Match 7
SC_R_MATCH_8	162	HSIO	0	Match 8
SC_R_MATCH_9	163	HSIO	0	Match 9
SC_R_MCLK_OUT_0	495	ADMA	0	MCLK out 0
SC_R_MCLK_OUT_1	496	ADMA	0	MCLK out 1
SC_R_MIPI_0	393	MIPI	0	MIPI
SC_R_MIPI_0_I2C_0	395	MIPI	0	I2C 0
SC_R_MIPI_0_I2C_1	396	MIPI	0	I2C 1
SC_R_MIPI_0_PWM_0	394	MIPI	0	PWM
SC_R_MIPI_1	397	MIPI	1	MIPI
SC_R_MIPI_1_I2C_0	399	MIPI	1	I2C 0
SC_R_MIPI_1_I2C_1	400	MIPI	1	I2C 1
SC_R_MIPI_1_PWM_0	398	MIPI	1	PWM
SC_R_MJPEG_DEC_MP	532	IMG	0	MJPEG decoder MP
SC_R_MJPEG_DEC_S0	385	IMG	0	MJPEG decoder stream 0
SC_R_MJPEG_DEC_S1	386	IMG	0	MJPEG decoder stream 1
SC_R_MJPEG_DEC_S2	387	IMG	0	MJPEG decoder stream 2
SC_R_MJPEG_DEC_S3	388	IMG	0	MJPEG decoder stream 3
SC_R_MJPEG_ENC_MP	533	IMG	0	MJPEG encoder MP
SC_R_MJPEG_ENC_S0	389	IMG	0	MJPEG encoder stream 0
SC_R_MJPEG_ENC_S1	390	IMG	0	MJPEG encoder stream 1
SC_R_MJPEG_ENC_S2	391	IMG	0	MJPEG encoder stream 2
SC_R_MJPEG_ENC_S3	392	IMG	0	MJPEG encoder stream 3
SC_R_MLB_0	253	CONN	0	MLB
SC_R_MQS_0	459	ADMA	0	MQS
SC_R_MU_0A	213	LSIO	0	Message Unit (MU) 0A
SC_R_MU_10A	223	LSIO	0	MU 10A
SC_R_MU_10B	232	LSIO	0	MU 10B
SC_R_MU_11A	224	LSIO	0	MU 11A
SC_R_MU_11B	233	LSIO	0	MU 11B
SC_R_MU_12A	225	LSIO	0	MU 12A
SC_R_MU_12B	234	LSIO	0	MU 12B
SC_R_MU_13A	226	LSIO	0	MU 13A
SC_R_MU_13B	235	LSIO	0	MU 13B
SC_R_MU_1A	214	LSIO	0	MU 1A

Resource	Index	Subsystem	Instance	Description
SC_R_MU_2A	215	LSIO	0	MU 2A
SC_R_MU_3A	216	LSIO	0	MU 3A
SC_R_MU_4A	217	LSIO	0	MU 4A
SC_R_MU_5A	218	LSIO	0	MU 5A
SC_R_MU_5B	227	LSIO	0	MU 5B
SC_R_MU_6A	219	LSIO	0	MU 6A
SC_R_MU_6B	228	LSIO	0	MU 6B
SC_R_MU_7A	220	LSIO	0	MU 7A
SC_R_MU_7B	229	LSIO	0	MU 7B
SC_R_MU_8A	221	LSIO	0	MU 8A
SC_R_MU_8B	230	LSIO	0	MU 8B
SC_R_MU_9A	222	LSIO	0	MU 9A
SC_R_MU_9B	231	LSIO	0	MU 9B
SC_R_NAND	265	CONN	0	NAND BCH/GPMI/DMA
SC_R_OCRAM	324	LSIO	0	OCRAM
SC_R_OTP	357	SC	0	OTP
SC_R_PCIE_B	169	HSIO	0	PCIE B
SC_R_PI_0	326	PI	0	Parallel Capture Interface
SC_R_PI_0_I2C_0	329	PI	0	I2C
SC_R_PI_0_PLL	330	PI	0	PLL
SC_R_PI_0_PWM_0	327	PI	0	PWM 0
SC_R_PMIC_0	497	BOARD	0	PMIC 0
SC_R_PMIC_1	498	BOARD	0	PMIC 1
SC_R_PMIC_2	521	BOARD	0	PMIC 2
SC_R_PWM_0	191	LSIO	0	Pulse Width Modulator (PWM) 0
SC_R_PWM_1	192	LSIO	0	PWM 1
SC_R_PWM_2	193	LSIO	0	PWM 2
SC_R_PWM_3	194	LSIO	0	PWM 3
SC_R_PWM_4	195	LSIO	0	PWM 4
SC_R_PWM_5	196	LSIO	0	PWM 5
SC_R_PWM_6	197	LSIO	0	PWM 6
SC_R_PWM_7	198	LSIO	0	PWM 7
SC_R_SAI_0	318	ADMA	0	SAI 0
SC_R_SAI_1	319	ADMA	0	SAI 1
SC_R_SAI_2	320	ADMA	0	SAI 2
SC_R_SAI_3	418	ADMA	0	SAI 3
SC_R_SAI_4	419	ADMA	0	SAI_4
SC_R_SAI_5	420	ADMA	0	SAI_5
SC_R_SC_I2C	345	SC	0	I2C
SC_R_SC_MU_0A0	347	SC	0	MU 0A0
SC_R_SC_MU_0A1	348	SC	0	MU 0A1
SC_R_SC_MU_0A2	349	SC	0	MU 0A2
SC_R_SC_MU_0A3	350	SC	0	MU 0A3
SC_R_SC_MU_0B	346	SC	0	MU 0B
SC_R_SC_MU_1A	351	SC	0	MU 1A
SC_R_SC_PID0	336	SC	0	PID0
SC_R_SC_PID1	337	SC	0	PID1

Resource	Index	Subsystem	Instance	Description
SC_R_SC_PID2	338	SC	0	PID2
SC_R_SC_PID3	339	SC	0	PID3
SC_R_SC_PID4	340	SC	0	PID4
SC_R_SC_PIT	343	SC	0	PIT
SC_R_SC_SEMA42	341	SC	0	SEMA42
SC_R_SC_TPM	342	SC	0	TPM
SC_R_SC_UART	344	SC	0	UART
SC_R_SDHC_0	248	CONN	0	uSDHC 1
SC_R_SDHC_1	249	CONN	0	uSDHC 2
SC_R_SDHC_2	250	CONN	0	uSDHC 3
SC_R_SECO	499	SC	0	SECO
SC_R_SECO_MU_2	503	SC	0	SECO MU 2
SC_R_SECO_MU_3	504	SC	0	SECO MU 3
SC_R_SECO_MU_4	505	SC	0	SECO MU 4
SC_R_SECVIO	44	SC	0	Security Violation
SC_R_SERDES_1	171	HSIO	0	PHYx1
SC_R_SPDIF_0	416	ADMA	0	SPDIF 0
SC_R_SPI_0	53	ADMA	0	SPI 0
SC_R_SPI_1	54	ADMA	0	SPI 1
SC_R_SPI_2	55	ADMA	0	SPI 2
SC_R_SPI_3	56	ADMA	0	SPI 3
SC_R_SYSCNT_CMP	353	SC	0	SYSCNT CMP
SC_R_SYSCNT_RD	352	SC	0	SYSCNT RD
SC_R_SYSTEM	355	SC	0	System
SC_R_UART_0	57	ADMA	0	UART 0
SC_R_UART_1	58	ADMA	0	UART 1
SC_R_UART_2	59	ADMA	0	UART 2
SC_R_UART_3	60	ADMA	0	UART 3
SC_R_USB_0	259	CONN	0	USB2.0 OTG
SC_R_USB_0_PHY	261	CONN	0	USB2.0 OTG phy
SC_R_USB_1	260	CONN	0	USB2.0 HSIC
SC_R_USB_2	262	CONN	0	USB3.0
SC_R_USB_2_PHY	263	CONN	0	USB3.0 phy
SC_R_VPU	540	VPU	0	VPU (XMEM peripheral)
SC_R_VPU_DEC_0	517	VPU	0	Decoder
SC_R_VPU_ENC_0	518	VPU	0	Encoder 0
SC_R_VPU_MU_0	535	VPU	0	MU 0
SC_R_VPU_MU_1	536	VPU	0	MU 1
SC_R_VPU_PID0	358	VPU	0	XMEM master ID=0
SC_R_VPU_PID1	359	VPU	0	XMEM master ID=1
SC_R_VPU_PID2	360	VPU	0	XMEM master ID=2
SC_R_VPU_PID3	361	VPU	0	XMEM master ID=3
SC_R_VPU_PID4	362	VPU	0	XMEM master ID=4
SC_R_VPU_PID5	363	VPU	0	XMEM master ID=5
SC_R_VPU_PID6	364	VPU	0	XMEM master ID=6
SC_R_VPU_PID7	365	VPU	0	XMEM master ID=7

Chapter 7

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

[PM: Power Management Service](#)

Resource	Clock	Default Rate	Set	Enable	Description
SC_R_A35	SC_PM_CLK_CPU	1200MHZ	Y		CPU
SC_R_ADC_0	SC_PM_CLK_PER		Y	Y	ADC 0 peripheral
SC_R_AUDIO_PLL_0	SC_PM_CLK_MISC		Y	Y	User programmable PLL
SC_R_AUDIO_PLL_0	SC_PM_CLK_MST_B↔ US		Y	Y	Audio rec 0
SC_R_AUDIO_PLL_0	SC_PM_CLK_SLV_B↔ US		Y	Y	Audio PLL div 0
SC_R_AUDIO_PLL_1	SC_PM_CLK_MISC		Y	Y	User programmable PLL
SC_R_AUDIO_PLL_1	SC_PM_CLK_MST_B↔ US		Y	Y	Audio rec 1
SC_R_AUDIO_PLL_1	SC_PM_CLK_SLV_B↔ US		Y	Y	Audio PLL div 1
SC_R_CAN_0	SC_PM_CLK_PER		Y	Y	CAN 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_DC_0	SC_PM_CLK_MISC0		Y	Y	Display 0
SC_R_DC_0	SC_PM_CLK_MISC1		Y	Y	Display 1
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

Resource	Clock	Default Rate	Set	Enable	Description
SC_R_DRC_0	SC_PM_CLK_SLV_B↔ US	600MHZ	Y		DRC root
SC_R_ELCDIF_PLL	SC_PM_CLK_MISC		Y	Y	eLCDIF PLL
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_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_LCD_0	SC_PM_CLK_MISC	24MHZ	Y	Y	Bypass
SC_R_LCD_0	SC_PM_CLK_PER		Y	Y	eLCDIF baud
SC_R_LCD_0	SC_PM_CLK_SLV_B↔ US		Y	Y	Pixel Link Mux
SC_R_LCD_0_PWM↔ _0	SC_PM_CLK_PER		Y	Y	PWM 0 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_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_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_MIPI_0	SC_PM_CLK_BYPASS		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		Y	Y	DPI - pixel
SC_R_MIPI_0	SC_PM_CLK_PHY	27MHZ	Y	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	24MHZ	Y	Y	I2C 0 baud
SC_R_MIPI_0_I2C_1	SC_PM_CLK_PER	24MHZ	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		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		Y	Y	DPI - pixel
SC_R_MIPI_1	SC_PM_CLK_PHY	27MHZ	Y	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	24MHZ	Y	Y	I2C 0 baud
SC_R_MIPI_1_I2C_1	SC_PM_CLK_PER	24MHZ	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_PI_0	SC_PM_CLK_BYPASS	24MHZ	Y	Y	Bypass
SC_R_PI_0	SC_PM_CLK_MISC0	24MHZ	Y	Y	MCLK
SC_R_PI_0	SC_PM_CLK_PER		Y	Y	Pixel
SC_R_PI_0_I2C_0	SC_PM_CLK_PER	24MHZ	Y	Y	I2C baud
SC_R_PI_0_PLL	SC_PM_CLK_PLL	960MHZ	Y	Y	User programmable PLL
SC_R_PI_0_PWM_0	SC_PM_CLK_PER	24MHZ	Y	Y	PWM baud
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

Resource	Clock	Default Rate	Set	Enable	Description
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
SC_R_UART_3	SC_PM_CLK_PER		Y	Y	UART 3 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

Chapter 8

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\(\)](#). Temp sensors can be accessed via these but use raw sensor values. The [sc_misc_set_temp\(\)](#) and [sc_misc_get_temp\(\)](#) functions use degrees in celsius. The list below can also be used to identify which resources have an associated temp sensor for these functions.

Note the control define values are defined in [types.h](#).

See also

[MISC: Miscellaneous Service](#)

Resource	Control	Width	Set	Description
SC_R_CAN_0	SC_C_IPG_DOZE	1	Y	CAN0 IPG_DOZE
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_DOZE	1	Y	CAN1 IPG_DOZE
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_DOZE	1	Y	CAN2 IPG_DOZE
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_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

Resource	Control	Width	Set	Description
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_DRC_0	SC_C_CALIB0	0	Y	Call DQS2DQ init function (for stress test only)
SC_R_DRC_0	SC_C_CALIB1	0	Y	Call RDBI bit deskew function (for stress test only)
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
SC_R_DRC_0	SC_C_TEMP_LOW	16	Y	Temperature sensor low limit alarm value
SC_R_DSP	SC_C_DSP_BYP	1	Y	DSP to DMA bypass (0=enb/1=dis)
SC_R_DSP	SC_C_OFS_AUDIO	8	Y	System address offset of AUDIO
SC_R_DSP	SC_C_OFS_AUDIO_ALT	8	Y	System alt address offset of AUDIO
SC_R_DSP	SC_C_OFS_IRQ	8	Y	System address offset of IRQ
SC_R_DSP	SC_C_OFS_PERIPH	8	Y	System address offset of PERIPH
SC_R_DSP	SC_C_OFS_SEL	1	Y	System address offset source select
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_FSPI_0	SC_C_MISC0	9	Y	FSPI 0 swap
SC_R_FSPI_0	SC_C_MISC1	9	Y	FSPI 0 swap mask
SC_R_GPU_0_PID0	SC_C_ID	4	Y	Identifier
SC_R_MIPI_0	SC_C_DPI_RESET	1	Y	MIPI DSI DPI reset

Resource	Control	Width	Set	Description
SC_R_MIPI_0	SC_C_DUAL_MODE	1	Y	LVDS SS single/dual
SC_R_MIPI_0	SC_C_MIPI_RESET	1	Y	MIPI DSI escape reset
SC_R_MIPI_0	SC_C_MODE	1	Y	MIPI-DSI / LVDS mode
SC_R_MIPI_0	SC_C_PHY_RESET	1	Y	MIPI DSI byte reset (PHY reset)
SC_R_MIPI_0	SC_C_PXL_LINK_RATE_CORRECTION	2	Y	Slave pixel link rate correction
SC_R_MIPI_0	SC_C_PXL_LINK_SEL	1	Y	MIPI-DSI / LVDS pixel link select
SC_R_MIPI_1	SC_C_DPI_RESET	1	Y	MIPI DSI DPI reset
SC_R_MIPI_1	SC_C_DUAL_MODE	1	Y	LVDS SS single/dual
SC_R_MIPI_1	SC_C_MIPI_RESET	1	Y	MIPI DSI escape reset
SC_R_MIPI_1	SC_C_MODE	1	Y	MIPI-DSI / LVDS mode
SC_R_MIPI_1	SC_C_PHY_RESET	1	Y	MIPI DSI byte reset (PHY reset)
SC_R_MIPI_1	SC_C_PXL_LINK_RATE_CORRECTION	2	Y	Slave pixel link rate correction
SC_R_MIPI_1	SC_C_PXL_LINK_SEL	1	Y	MIPI-DSI / LVDS pixel link select
SC_R_MLB_0	SC_C_CLKDIV	3	Y	MLB PLL Multiplier (0 - refclk, 1, 2 and 4)
SC_R_MLB_0	SC_C_MODE	1	Y	0 - disable MLB PLL, 1 - Enable MLB PLL
SC_R_MLB_0	SC_C_VOLTAGE	1	Y	0 - disable MLB PHY BG, 1 - Enable MLB PHY BG (Required for MLB 6-pin)
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

Chapter 9

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.

Note the pad define values (index) are listed here but the define should be used. These are defined in the SoC-specific [pads.h](#). Functions like [sc_rm_set_pad_movable\(\)](#) take a pad range. This range is numerical by index and not alphabetical by define name. Range should be lowest to highest index.

See also

[PAD: Pad Service](#)

Pad	Index	Mux Options
SC_P_ADC_IN0	100	ADMA.ADC.IN0, M40.I2C0.SCL, M40.GPIO0.IO00, LSIO.GPIO1.IO10
SC_P_ADC_IN1	99	ADMA.ADC.IN1, M40.I2C0.SDA, M40.GPIO0.IO01, LSIO.GPIO1.IO09
SC_P_ADC_IN2	102	ADMA.ADC.IN2, M40.UART0.RX, M40.GPIO0.IO02, ADMA.ACM.MCLK_IN0, LSIO.GPIO1.IO12
SC_P_ADC_IN3	101	ADMA.ADC.IN3, M40.UART0.TX, M40.GPIO0.IO03, ADMA.ACM.MCLK_OUT0, LSIO.GPIO1.IO11
SC_P_ADC_IN4	104	ADMA.ADC.IN4, M40.TPM0.CH0, M40.GPIO0.IO04, LSIO.GPIO1.IO14
SC_P_ADC_IN5	103	ADMA.ADC.IN5, M40.TPM0.CH1, M40.GPIO0.IO05, LSIO.GPIO1.IO13
SC_P_COMP_CTL_GPIO_1V8_3V3_ENET_EN↔ETB0	43	
SC_P_COMP_CTL_GPIO_1V8_3V3_ENET_EN↔ETB1	50	

Pad	Index	Mux Options
SC_P_COMP_CTL_GPIO_1V8_3V3_GPI OCT	54	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIOLH	115	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHB	68	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHD	150	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHK	81	
SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHT	98	
SC_P_COMP_CTL_GPIO_1V8_3V3_MIPIDSIG↔ PIO	124	
SC_P_COMP_CTL_GPIO_1V8_3V3_PCIESEP	3	
SC_P_COMP_CTL_GPIO_1V8_3V3_QSPI0A	164	
SC_P_COMP_CTL_GPIO_1V8_3V3_QSPI0B	173	
SC_P_COMP_CTL_GPIO_1V8_3V3_SD1FIX0	15	
SC_P_COMP_CTL_GPIO_1V8_3V3_SD1FIX1	22	
SC_P_COMP_CTL_GPIO_1V8_3V3_VSEL3	36	
SC_P_COMP_CTL_GPIO_1V8_3V3_VSELSEP	29	
SC_P_COMP_CTL_GPIO_3V3_USB3IO	8	
SC_P_CSI_D00	136	CI_PI.D02, ADMA.SAI0.RXC
SC_P_CSI_D01	137	CI_PI.D03, ADMA.SAI0.RXD
SC_P_CSI_D02	138	CI_PI.D04, ADMA.SAI0.RXFS
SC_P_CSI_D03	139	CI_PI.D05, ADMA.SAI2.RXC
SC_P_CSI_D04	140	CI_PI.D06, ADMA.SAI2.RXD
SC_P_CSI_D05	141	CI_PI.D07, ADMA.SAI2.RXFS
SC_P_CSI_D06	142	CI_PI.D08, ADMA.SAI3.RXC
SC_P_CSI_D07	143	CI_PI.D09, ADMA.SAI3.RXD
SC_P_CSI_EN	148	CI_PI.EN, CI_PI.I2C.SCL, ADMA.I2C3.SCL, AD↔ MA.SPI1.SDI, LSIO.GPIO3.IO02
SC_P_CSI_HSYNC	144	CI_PI.HSYNC, CI_PI.D00, ADMA.SAI3.RXFS
SC_P_CSI_MCLK	147	CI_PI.MCLK, MIPI_CSI0.I2C0.SDA, ADMA.SPI1.↔ SDO, LSIO.GPIO3.IO01
SC_P_CSI_PCLK	146	CI_PI.PCLK, MIPI_CSI0.I2C0.SCL, ADMA.SPI1.↔ SCK, LSIO.GPIO3.IO00
SC_P_CSI_RESET	149	CI_PI.RESET, CI_PI.I2C.SDA, ADMA.I2C3.SDA, ADMA.SPI1.CS0, LSIO.GPIO3.IO03
SC_P_CSI_VSYNC	145	CI_PI.VSYNC, CI_PI.D01
SC_P_CTL_NAND_DQS_P_N	28	
SC_P_CTL_NAND_RE_P_N	25	
SC_P_EMMC0_CLK	9	CONN.EMMC0.CLK, CONN.NAND.READY_B, L↔ SIO.GPIO4.IO07
SC_P_EMMC0_CMD	10	CONN.EMMC0.CMD, CONN.NAND.DQS, LSIO.↔ GPIO4.IO08
SC_P_EMMC0_DATA0	11	CONN.EMMC0.DATA0, CONN.NAND.DATA00, LSIO.GPIO4.IO09
SC_P_EMMC0_DATA1	12	CONN.EMMC0.DATA1, CONN.NAND.DATA01, LSIO.GPIO4.IO10
SC_P_EMMC0_DATA2	13	CONN.EMMC0.DATA2, CONN.NAND.DATA02, LSIO.GPIO4.IO11
SC_P_EMMC0_DATA3	14	CONN.EMMC0.DATA3, CONN.NAND.DATA03, LSIO.GPIO4.IO12

Pad	Index	Mux Options
SC_P_EMMC0_DATA4	16	CONN.EMMC0.DATA4, CONN.NAND.DATA04, CONN.EMMC0.WP, LSIO.GPIO4.IO13
SC_P_EMMC0_DATA5	17	CONN.EMMC0.DATA5, CONN.NAND.DATA05, CONN.EMMC0.VSELECT, LSIO.GPIO4.IO14
SC_P_EMMC0_DATA6	18	CONN.EMMC0.DATA6, CONN.NAND.DATA06, CONN.MLB.CLK, LSIO.GPIO4.IO15
SC_P_EMMC0_DATA7	19	CONN.EMMC0.DATA7, CONN.NAND.DATA07, CONN.MLB.SIG, LSIO.GPIO4.IO16
SC_P_EMMC0_RESET_B	21	CONN.EMMC0.RESET_B, CONN.NAND.WP_B, LSIO.GPIO4.IO18
SC_P_EMMC0_STROBE	20	CONN.EMMC0.STROBE, CONN.NAND.CLE, CONN.MLB.DATA, LSIO.GPIO4.IO17
SC_P_ENET0_MDC	53	CONN.ENET0.MDC, ADMA.I2C3.SCL, CONN.ENET1.MDC, LSIO.GPIO5.IO11
SC_P_ENET0_MDIO	52	CONN.ENET0.MDIO, ADMA.I2C3.SDA, CONN.ENET1.MDIO, LSIO.GPIO5.IO10
SC_P_ENET0_REFCLK_125M_25M	51	CONN.ENET0.REFCLK_125M_25M, CONN.ENET0.PPS, CONN.ENET1.PPS, LSIO.GPIO5.IO09
SC_P_ENET0_RGMII_RXC	44	CONN.ENET0.RGMII_RXC, CONN.MLB.DATTA, CONN.NAND.WE_B, CONN.USDHC1.CLK, LSIO.GPIO5.IO03
SC_P_ENET0_RGMII_RXD0	46	CONN.ENET0.RGMII_RXD0, CONN.USDHC1.DATTA0, LSIO.GPIO5.IO05
SC_P_ENET0_RGMII_RXD1	47	CONN.ENET0.RGMII_RXD1, CONN.USDHC1.DATTA1, LSIO.GPIO5.IO06
SC_P_ENET0_RGMII_RXD2	48	CONN.ENET0.RGMII_RXD2, CONN.ENET0.RMII_RX_ER, CONN.USDHC1.DATA2, LSIO.GPIO5.IO07
SC_P_ENET0_RGMII_RXD3	49	CONN.ENET0.RGMII_RXD3, CONN.NAND.ALE, CONN.USDHC1.DATA3, LSIO.GPIO5.IO08
SC_P_ENET0_RGMII_RX_CTL	45	CONN.ENET0.RGMII_RX_CTL, CONN.USDHC1.CMD, LSIO.GPIO5.IO04
SC_P_ENET0_RGMII_TXC	37	CONN.ENET0.RGMII_TXC, CONN.ENET0.RCLK50M_OUT, CONN.ENET0.RCLK50M_IN, CONN.NAND.CE1_B, LSIO.GPIO4.IO29
SC_P_ENET0_RGMII_TXD0	39	CONN.ENET0.RGMII_TXD0, CONN.USDHC1.VSELECT, LSIO.GPIO4.IO31
SC_P_ENET0_RGMII_TXD1	40	CONN.ENET0.RGMII_TXD1, CONN.USDHC1.WP, LSIO.GPIO5.IO00
SC_P_ENET0_RGMII_TXD2	41	CONN.ENET0.RGMII_TXD2, CONN.MLB.CLK, CONN.NAND.CE0_B, CONN.USDHC1.CD_B, LSIO.GPIO5.IO01
SC_P_ENET0_RGMII_TXD3	42	CONN.ENET0.RGMII_TXD3, CONN.MLB.SIG, CONN.NAND.RE_B, LSIO.GPIO5.IO02
SC_P_ENET0_RGMII_TX_CTL	38	CONN.ENET0.RGMII_TX_CTL, CONN.USDHC1.RESET_B, LSIO.GPIO4.IO30
SC_P_ESAI0_FSR	55	ADMA.ESAI0.FSR, CONN.ENET1.RCLK50M_OUT, ADMA.LCDIF.D00, CONN.ENET1.RGMII_TXC, CONN.ENET1.RCLK50M_IN

Pad	Index	Mux Options
SC_P_ESAI0_FST	56	ADMA.ESAI0.FST, CONN.MLB.CLK, ADMA.LC↔ DIF.D01, CONN.ENET1.RGMII_TXD2, LSIO.GP↔ IO0.IO01
SC_P_ESAI0_SCKR	57	ADMA.ESAI0.SCKR, ADMA.LCDIF.D02, CONN.↔ ENET1.RGMII_TX_CTL, LSIO.GPIO0.IO02
SC_P_ESAI0_SCKT	58	ADMA.ESAI0.SCKT, CONN.MLB.SIG, ADMA.LC↔ DIF.D03, CONN.ENET1.RGMII_TXD3, LSIO.GPI↔ O0.IO03
SC_P_ESAI0_TX0	59	ADMA.ESAI0.TX0, CONN.MLB.DATA, ADMA.LC↔ DIF.D04, CONN.ENET1.RGMII_RXC, LSIO.GPI↔ O0.IO04
SC_P_ESAI0_TX1	60	ADMA.ESAI0.TX1, ADMA.LCDIF.D05, CONN.E↔ NET1.RGMII_RXD3, LSIO.GPIO0.IO05
SC_P_ESAI0_TX2_RX3	61	ADMA.ESAI0.TX2_RX3, CONN.ENET1.RMII_R↔ X_ER, ADMA.LCDIF.D06, CONN.ENET1.RGMII↔ _RXD2, LSIO.GPIO0.IO06
SC_P_ESAI0_TX3_RX2	62	ADMA.ESAI0.TX3_RX2, ADMA.LCDIF.D07, CO↔ NN.ENET1.RGMII_RXD1, LSIO.GPIO0.IO07
SC_P_ESAI0_TX4_RX1	63	ADMA.ESAI0.TX4_RX1, ADMA.LCDIF.D08, CO↔ NN.ENET1.RGMII_TXD0, LSIO.GPIO0.IO08
SC_P_ESAI0_TX5_RX0	64	ADMA.ESAI0.TX5_RX0, ADMA.LCDIF.D09, CO↔ NN.ENET1.RGMII_TXD1, LSIO.GPIO0.IO09
SC_P_FLEXCAN0_RX	105	ADMA.FLEXCAN0.RX, ADMA.SAI2.RXC, ADM↔ A.UART0.RTS_B, ADMA.SAI1.TXC, LSIO.GPI↔ O1.IO15
SC_P_FLEXCAN0_TX	106	ADMA.FLEXCAN0.TX, ADMA.SAI2.RXD, ADM↔ A.UART0.CTS_B, ADMA.SAI1.TXFS, LSIO.GPI↔ O1.IO16
SC_P_FLEXCAN1_RX	107	ADMA.FLEXCAN1.RX, ADMA.SAI2.RXFS, ADM↔ A.FTM.CH2, ADMA.SAI1.TXD, LSIO.GPIO1.IO17
SC_P_FLEXCAN1_TX	108	ADMA.FLEXCAN1.TX, ADMA.SAI3.RXC, ADM↔ A.DMA0.REQ_IN0, ADMA.SAI1.RXD, LSIO.GPI↔ O1.IO18
SC_P_FLEXCAN2_RX	109	ADMA.FLEXCAN2.RX, ADMA.SAI3.RXD, ADM↔ A.UART3.RX, ADMA.SAI1.RXFS, LSIO.GPIO1.I↔ O19
SC_P_FLEXCAN2_TX	110	ADMA.FLEXCAN2.TX, ADMA.SAI3.RXFS, ADM↔ A.UART3.TX, ADMA.SAI1.RXC, LSIO.GPIO1.IO20
SC_P_JTAG_TRST_B	125	SCU.JTAG.TRST_B, SCU.WDOG0.WDOG_OUT
SC_P_MCLK_IN0	75	ADMA.ACM.MCLK_IN0, ADMA.ESAI0.RX_HF_↔ CLK, ADMA.LCDIF.VSYNC, ADMA.SPI2.SDI, LS↔ IO.GPIO0.IO19
SC_P_MCLK_IN1	74	ADMA.ACM.MCLK_IN1, ADMA.I2C3.SDA, ADM↔ A.LCDIF.EN, ADMA.SPI2.SCK, ADMA.LCDIF.D17
SC_P_MCLK_OUT0	76	ADMA.ACM.MCLK_OUT0, ADMA.ESAI0.TX_HF_↔ _CLK, ADMA.LCDIF.CLK, ADMA.SPI2.SDO, LSI↔ O.GPIO0.IO20
SC_P_MIPI_CSI0_GPIO0_00	155	MIPI_CSI0.GPIO0.IO00, ADMA.I2C0.SCL, LSIO.↔ GPIO3.IO08
SC_P_MIPI_CSI0_GPIO0_01	154	MIPI_CSI0.GPIO0.IO01, ADMA.I2C0.SDA, LSIO.↔ GPIO3.IO07

Pad	Index	Mux Options
SC_P_MIPI_CSI0_I2C0_SCL	152	MIPI_CSI0.I2C0_SCL, MIPI_CSI0.GPIO0.IO02, L↔SIO.GPIO3.IO05
SC_P_MIPI_CSI0_I2C0_SDA	153	MIPI_CSI0.I2C0_SDA, MIPI_CSI0.GPIO0.IO03, L↔SIO.GPIO3.IO06
SC_P_MIPI_CSI0_MCLK_OUT	151	MIPI_CSI0.ACM.MCLK_OUT, LSIO.GPIO3.IO04
SC_P_MIPI_DSI0_GPIO0_00	118	MIPI_DSI0.GPIO0.IO00, ADMA.I2C1_SCL, MIPI↔_DSI0.PWM0.OUT, LSIO.GPIO1.IO27
SC_P_MIPI_DSI0_GPIO0_01	119	MIPI_DSI0.GPIO0.IO01, ADMA.I2C1_SDA, LSIO.↔GPIO1.IO28
SC_P_MIPI_DSI0_I2C0_SCL	116	MIPI_DSI0.I2C0_SCL, MIPI_DSI1.GPIO0.IO02, L↔SIO.GPIO1.IO25
SC_P_MIPI_DSI0_I2C0_SDA	117	MIPI_DSI0.I2C0_SDA, MIPI_DSI1.GPIO0.IO03, L↔SIO.GPIO1.IO26
SC_P_MIPI_DSI1_GPIO0_00	122	MIPI_DSI1.GPIO0.IO00, ADMA.I2C2_SCL, MIPI↔_DSI1.PWM0.OUT, LSIO.GPIO1.IO31
SC_P_MIPI_DSI1_GPIO0_01	123	MIPI_DSI1.GPIO0.IO01, ADMA.I2C2_SDA, LSIO.↔GPIO2.IO00
SC_P_MIPI_DSI1_I2C0_SCL	120	MIPI_DSI1.I2C0_SCL, MIPI_DSI0.GPIO0.IO02, L↔SIO.GPIO1.IO29
SC_P_MIPI_DSI1_I2C0_SDA	121	MIPI_DSI1.I2C0_SDA, MIPI_DSI0.GPIO0.IO03, L↔SIO.GPIO1.IO30
SC_P_PCIE_CTRL0_CLKREQ_B	1	HSIO.PCIE0.CLKREQ_B, LSIO.GPIO4.IO01
SC_P_PCIE_CTRL0_PERST_B	0	HSIO.PCIE0.PERST_B, LSIO.GPIO4.IO00
SC_P_PCIE_CTRL0_WAKE_B	2	HSIO.PCIE0.WAKE_B, LSIO.GPIO4.IO02
SC_P_PMIC_I2C_SCL	126	SCU.PMIC_I2C_SCL, SCU.GPIO0.IOXX_PMIC_↔A35_ON, LSIO.GPIO2.IO01
SC_P_PMIC_I2C_SDA	127	SCU.PMIC_I2C_SDA, SCU.GPIO0.IOXX_PMIC_↔GPU_ON, LSIO.GPIO2.IO02
SC_P_PMIC_INT_B	128	SCU.DSC.PMIC_INT_B
SC_P_QSPI0A_DATA0	156	LSIO.QSPI0A.DATA0, LSIO.GPIO3.IO09
SC_P_QSPI0A_DATA1	157	LSIO.QSPI0A.DATA1, LSIO.GPIO3.IO10
SC_P_QSPI0A_DATA2	158	LSIO.QSPI0A.DATA2, LSIO.GPIO3.IO11
SC_P_QSPI0A_DATA3	159	LSIO.QSPI0A.DATA3, LSIO.GPIO3.IO12
SC_P_QSPI0A_DQS	160	LSIO.QSPI0A.DQS, LSIO.GPIO3.IO13
SC_P_QSPI0A_SCLK	163	LSIO.QSPI0A.SCLK, LSIO.GPIO3.IO16
SC_P_QSPI0A_SS0_B	161	LSIO.QSPI0A.SS0_B, LSIO.GPIO3.IO14
SC_P_QSPI0A_SS1_B	162	LSIO.QSPI0A.SS1_B, LSIO.GPIO3.IO15
SC_P_QSPI0B_DATA0	166	LSIO.QSPI0B.DATA0, LSIO.QSPI1A.DATA0, LS↔IO.KPP0.COL1, LSIO.GPIO3.IO18
SC_P_QSPI0B_DATA1	167	LSIO.QSPI0B.DATA1, LSIO.QSPI1A.DATA1, LS↔IO.KPP0.COL2, LSIO.GPIO3.IO19
SC_P_QSPI0B_DATA2	168	LSIO.QSPI0B.DATA2, LSIO.QSPI1A.DATA2, LS↔IO.KPP0.COL3, LSIO.GPIO3.IO20
SC_P_QSPI0B_DATA3	169	LSIO.QSPI0B.DATA3, LSIO.QSPI1A.DATA3, LS↔IO.KPP0.ROW0, LSIO.GPIO3.IO21
SC_P_QSPI0B_DQS	170	LSIO.QSPI0B.DQS, LSIO.QSPI1A.DQS, LSIO.K↔PP0.ROW1, LSIO.GPIO3.IO22
SC_P_QSPI0B_SCLK	165	LSIO.QSPI0B.SCLK, LSIO.QSPI1A.SCLK, LSIO.↔KPP0.COL0, LSIO.GPIO3.IO17

Pad	Index	Mux Options
SC_P_QSPI0B_SS0_B	171	LSIO.QSPI0B.SS0_B, LSIO.QSPI1A.SS0_B, LSI↔O.KPP0.ROW2, LSIO.GPIO3.IO23
SC_P_QSPI0B_SS1_B	172	LSIO.QSPI0B.SS1_B, LSIO.QSPI1A.SS1_B, LSI↔O.KPP0.ROW3, LSIO.GPIO3.IO24
SC_P_SAI0_RXD	84	ADMA.SAI0.RXD, ADMA.SAI1.RXFS, ADMA.SP↔I1.CS0, ADMA.LCDIF.D20, LSIO.GPIO0.IO27
SC_P_SAI0_TXC	83	ADMA.SAI0.TXC, ADMA.SAI1.TXD, ADMA.SP↔I1.SDI, ADMA.LCDIF.D19, LSIO.GPIO0.IO26
SC_P_SAI0_TXD	82	ADMA.SAI0.TXD, ADMA.SAI1.RXC, ADMA.SP↔I1.SDO, ADMA.LCDIF.D18, LSIO.GPIO0.IO25
SC_P_SAI0_TXFS	85	ADMA.SAI0.TXFS, ADMA.SPI2.CS1, ADMA.SP↔I1.SCK, LSIO.GPIO0.IO28
SC_P_SAI1_RXC	87	ADMA.SAI1.RXC, ADMA.SAI1.TXC, ADMA.LCD↔IF.D22, LSIO.GPIO0.IO30
SC_P_SAI1_RXD	86	ADMA.SAI1.RXD, ADMA.SAI0.RXFS, ADMA.SP↔I1.CS1, ADMA.LCDIF.D21, LSIO.GPIO0.IO29
SC_P_SAI1_RXFS	88	ADMA.SAI1.RXFS, ADMA.SAI1.TXFS, ADMA.L↔CDIF.D23, LSIO.GPIO0.IO31
SC_P_SCU_BOOT_MODE0	132	SCU.DSC.BOOT_MODE0
SC_P_SCU_BOOT_MODE1	133	SCU.DSC.BOOT_MODE1
SC_P_SCU_BOOT_MODE2	134	SCU.DSC.BOOT_MODE2, SCU.PMIC_I2C.SDA
SC_P_SCU_BOOT_MODE3	135	SCU.DSC.BOOT_MODE3, SCU.PMIC_I2C.SCL, SCU.DSC.RTC_CLOCK_OUTPUT_32K
SC_P_SCU_GPIO0_00	129	SCU.GPIO0.IO00, SCU.UART0.RX, M40.UAR↔T0.RX, ADMA.UART3.RX, LSIO.GPIO2.IO03
SC_P_SCU_GPIO0_01	130	SCU.GPIO0.IO01, SCU.UART0.TX, M40.UAR↔T0.TX, ADMA.UART3.TX, SCU.WDOG0.WDOG↔_OUT
SC_P_SCU_PMIC_STANDBY	131	SCU.DSC.PMIC_STANDBY
SC_P_SPDIF0_EXT_CLK	67	ADMA.SPDIFF0.EXT_CLK, ADMA.LCDIF.D12, CONN.ENET1.REFCLK_125M_25M, LSIO.GPI↔O0.IO12
SC_P_SPDIF0_RX	65	ADMA.SPDIFF0.RX, ADMA.MQS.R, ADMA.LCD↔IF.D10, CONN.ENET1.RGMII_RXD0, LSIO.GPI↔O0.IO10
SC_P_SPDIF0_TX	66	ADMA.SPDIFF0.TX, ADMA.MQS.L, ADMA.LCDIF.↔D11, CONN.ENET1.RGMII_RX_CTL, LSIO.GPI↔O0.IO11
SC_P_SPI0_CS0	97	ADMA.SPI0.CS0, ADMA.SAI0.RXD, M40.TPM0.↔CH1, M40.GPIO0.IO03, LSIO.GPIO1.IO08
SC_P_SPI0_CS1	96	ADMA.SPI0.CS1, ADMA.SAI0.RXC, ADMA.SA↔I1.TXD, ADMA.LCD_PWM0.OUT, LSIO.GPIO1.I↔O07
SC_P_SPI0_SCK	93	ADMA.SPI0.SCK, ADMA.SAI0.TXC, M40.I2C0.S↔CL, M40.GPIO0.IO00, LSIO.GPIO1.IO04
SC_P_SPI0_SDI	94	ADMA.SPI0.SDI, ADMA.SAI0.TXD, M40.TPM0.C↔H0, M40.GPIO0.IO02, LSIO.GPIO1.IO05
SC_P_SPI0_SDO	95	ADMA.SPI0.SDO, ADMA.SAI0.TXFS, M40.I2C0.↔SDA, M40.GPIO0.IO01, LSIO.GPIO1.IO06
SC_P_SPI2_CS0	89	ADMA.SPI2.CS0, LSIO.GPIO1.IO00

Pad	Index	Mux Options
SC_P_SPI2_SCK	92	ADMA.SPI2.SCK, LSIO.GPIO1.IO03
SC_P_SPI2_SDI	91	ADMA.SPI2.SDI, LSIO.GPIO1.IO02
SC_P_SPI2_SDO	90	ADMA.SPI2.SDO, LSIO.GPIO1.IO01
SC_P_SPI3_CS0	72	ADMA.SPI3.CS0, ADMA.ACM.MCLK_OUT1, ADMA.LCDIF.HSYNC, LSIO.GPIO0.IO16
SC_P_SPI3_CS1	73	ADMA.SPI3.CS1, ADMA.I2C3.SCL, ADMA.LCDIF.RESET, ADMA.SPI2.CS0, ADMA.LCDIF.D16
SC_P_SPI3_SCK	69	ADMA.SPI3.SCK, ADMA.LCDIF.D13, LSIO.GPIO0.IO13
SC_P_SPI3_SDI	71	ADMA.SPI3.SDI, ADMA.LCDIF.D15, LSIO.GPIO0.IO15
SC_P_SPI3_SDO	70	ADMA.SPI3.SDO, ADMA.LCDIF.D14, LSIO.GPIO0.IO14
SC_P_UART0_RX	111	ADMA.UART0.RX, ADMA.MQS.R, ADMA.FLEXCAN0.RX, SCU.UART0.RX, LSIO.GPIO1.IO21
SC_P_UART0_TX	112	ADMA.UART0.TX, ADMA.MQS.L, ADMA.FLEXCAN0.TX, SCU.UART0.TX, LSIO.GPIO1.IO22
SC_P_UART1_CTS_B	80	ADMA.UART1.CTS_B, LSIO.PWM3.OUT, ADMA.LCDIF.D17, LSIO.GPT1.COMPARE, LSIO.GPIO0.IO24
SC_P_UART1_RTS_B	79	ADMA.UART1.RTS_B, LSIO.PWM2.OUT, ADMA.LCDIF.D16, LSIO.GPT1.CAPTURE, LSIO.GPT0.CLK
SC_P_UART1_RX	78	ADMA.UART1.RX, LSIO.PWM1.OUT, LSIO.GPT0.COMPARE, LSIO.GPT1.CLK, LSIO.GPIO0.IO22
SC_P_UART1_TX	77	ADMA.UART1.TX, LSIO.PWM0.OUT, LSIO.GPT0.CAPTURE, LSIO.GPIO0.IO21
SC_P_UART2_RX	114	ADMA.UART2.RX, ADMA.FTM.CH0, ADMA.FLEXCAN1.RX, LSIO.GPIO1.IO24
SC_P_UART2_TX	113	ADMA.UART2.TX, ADMA.FTM.CH1, ADMA.FLEXCAN1.TX, LSIO.GPIO1.IO23
SC_P_USB_SS3_TC0	4	ADMA.I2C1.SCL, CONN.USB_OTG1.PWR, CONN.USB_OTG2.PWR, LSIO.GPIO4.IO03
SC_P_USB_SS3_TC1	5	ADMA.I2C1.SCL, CONN.USB_OTG2.PWR, LSIO.GPIO4.IO04
SC_P_USB_SS3_TC2	6	ADMA.I2C1.SDA, CONN.USB_OTG1.OC, CONN.USB_OTG2.OC, LSIO.GPIO4.IO05
SC_P_USB_SS3_TC3	7	ADMA.I2C1.SDA, CONN.USB_OTG2.OC, LSIO.GPIO4.IO06
SC_P_USDHC1_CD_B	27	CONN.USDHC1.CD_B, CONN.NAND.DQS_P, ADMA.SPI2.CS0, CONN.NAND.DQS, LSIO.GPIO4.IO22
SC_P_USDHC1_CLK	30	CONN.USDHC1.CLK, ADMA.UART3.RX, LSIO.GPIO4.IO23
SC_P_USDHC1_CMD	31	CONN.USDHC1.CMD, CONN.NAND.CE0_B, ADMA.MQS.R, LSIO.GPIO4.IO24
SC_P_USDHC1_DATA0	32	CONN.USDHC1.DATA0, CONN.NAND.CE1_B, ADMA.MQS.L, LSIO.GPIO4.IO25
SC_P_USDHC1_DATA1	33	CONN.USDHC1.DATA1, CONN.NAND.RE_B, ADMA.UART3.TX, LSIO.GPIO4.IO26

Pad	Index	Mux Options
SC_P_USDHC1_DATA2	34	CONN.USDHC1.DATA2, CONN.NAND.WE_B, A↔DMA.UART3.CTS_B, LSIO.GPIO4.IO27
SC_P_USDHC1_DATA3	35	CONN.USDHC1.DATA3, CONN.NAND.ALE, AD↔MA.UART3.RTS_B, LSIO.GPIO4.IO28
SC_P_USDHC1_RESET_B	23	CONN.USDHC1.RESET_B, CONN.NAND.RE_N, ADMA.SPI2.SCK, LSIO.GPIO4.IO19
SC_P_USDHC1_VSELECT	24	CONN.USDHC1.VSELECT, CONN.NAND.RE_P, ADMA.SPI2.SDO, CONN.NAND.RE_B, LSIO.GP↔IO4.IO20
SC_P_USDHC1_WP	26	CONN.USDHC1.WP, CONN.NAND.DQS_N, AD↔MA.SPI2.SDI, LSIO.GPIO4.IO21

Chapter 10

RPC Protocol

Clients of the SCFW make function calls using a Remote Procedure Call (RPC) mechanism. This is constructed on top of an Inter-Processor Communication (IPC) layer. The RPC mechanism is independent of the IPC mechanism. All that is required is that the IPC mechanism can send and receive messages consisting of a series of 32-bit words. Message lengths can vary between different APIs but are fixed for a specific API.

The RPC protocol is documented here for reference ONLY. It is expected clients will make use of the exported API. Using the protocol directly is not supported and makes it impossible to provide customer support.

10.1 Remote Procedure Call

The RPC implementation is all generated via script. Interface Description Language (IDL) lines exist in the API header files to describe the calling conventions of each API call. This supports integer and unsigned values of 8-, 16-, 32-, and 64-bit sizes. Special provisions are also provided for boolean types and error returns. Float is not supported.

The RPC mechanism is synchronous. Almost all API calls construct a series of 32-bit words to form a request, send the request, receive a response, deconstruct the response (also a series of 32-bit words), and return to the caller. The corresponding function on the SCFW side (server side) is not called until a complete valid request is received.

Messages consist of a header followed by a variable number of parameter words. Parameter words are packed starting with the largest parameters. Parameters passed by pointer may exist in the send request and the response depending on how they are defined in the IDL (in, out, both). Empty slots are undefined and may not be 0.

The header is a single 32-bit word consisting of four bytes:

- **version** - the protocol version
- **size** - size of the message in words including the header
- **svc** - service index
- **func** - function index within the service

Note for return messages, svc=1U. Also, to minimize the number of words in the return value, func is replaced by the return value if it is an 8-bit type.

10.2 Inter-Processor Communication

The primary IPC mechanism implemented by the SCFW is via Message Unit (MU) IP (up to eight of them).

The MU has four TX and four RX registers, each capable of generating an interrupt. The SCFW IPC uses all four to create a single channel in both directions.

All message start with the first MU TX register. Words are written in order into successive registers and wrap back to the first if the message is greater than four words. Transmission blocks if the next TX register is not empty. The response is handled the same way. It always starts with the first RX register and wraps if the message is longer than four words. Transmission blocks if the next RX register is empty. On the SCFW-side, this is all interrupt driven. The message is buffered (per MU) and the API call into the SCFW only occurs when a complete valid request is received.

RPC messages can take many mS to complete. The API is fully synchronous and a timeout period should not be used as it would confuse the IPC state machine in the SCFW.

There is also an IPC mechanism implemented via UART as part of the SCFW debug monitor. This can be used to write test code that runs on a host connected via the SCU UART.

10.3 Interrupts

10.3.1 SCFW to Client Interrupts

The client-side IPC driver usually makes use of the MU TX/RX interrupts to transfer data. In addition, the SCFW makes use of the MU general-purpose interrupts to inform the client of various events that need attention. These four interrupts are defined in the `rpc.h` file delivered as part of the API export package:

Interrupt	MU GIRx	Use
SC_RPC_MU_GIR_SVC	0	Interrupt Service Request
SC_RPC_MU_GIR_WAKE	1	Cortex-M Wake
SC_RPC_MU_GIR_BOOT	2	Cold Boot Flag
SC_RPC_MU_GIR_DBG	3	Cortex-M Debug Wake

10.3.1.1 Interrupt Service Request

This interrupt is generated when the SCFW needs servicing. This behavior is managed using the [Interrupt Service API](#). Interrupts are collected into groups, each containing up to 32 interrupts. The groups and interrupts are listed in the `api.h` file for the interrupt service.

The client should call `sc_irq_enable()` to enable an interrupt. When the interrupt is serviced, the client should call `sc_irq_status()` for each group to determine the needed actions and figure out what component (i.e. OS driver) call-backs should be called. Calling `sc_irq_status()` also clears the pending status of the interrupt. As a result, the SCFW Interrupt Service acts as an interrupt collector/dispatcher mechanism.

10.3.1.2 Cortex-M Wake

This MU general-purpose IRQ is used to force a wakeup event prior to stopping a Cortex-M CPU in order to maintain coherency between the AWIC and core sleep state. Cortex-M SW should always enable.

10.3.1.3 Cold Boot Flag

This interrupt is not used as an interrupt. It is used as a clearable flag that indicates the client is booting after a cold reset. Because the SoC has multiple CPUs that could be sharing memory, a cold reset of a single CPU may not be able to actually reset the memory that CPU is using. This flag informs clients they should clear any state maintained in memory.

10.3.1.4 Cortex-M Debug Wake

This MU general-purpose IRQ is used to force a wakeup event prior to attaching debug to a Cortex-M CPU. This wake event will be sent when the respective CoreSight GPR (Granular Power Requestor) is set by the debugger to notify SCFW that power/clocks should be restored to Cortex-M core debug registers. Cortex-M SW should always enable.

10.3.2 Client to SCFW Interrupts

Interrupt	MU GIRx	Use
SC_RPC_MU_GIR_RST	0	RPC/IPC Reset Request

10.3.2.1 RPC/IPC Reset Request

This MU general-purpose IRQ is used to request the SCFW to reset the RPC/IPC interface. This request can be used to recover the RPC/IPC interface if a client driver/task is terminated/restarted in the middle of RPC/IPC communication. The RPC/IPC reset sequence should be executed as follows:

1. Driver or task performing RPC/IPC communication is stopped
2. Client requests RPC reset by writing to the SC_RPC_MU_GIR_RST bit of the MU.CR register
3. Client polls MU.CR register until SC_RPC_MU_GIR_RST bit is cleared to indicate SCFW has completed the RPC/IPC reset
4. Client side of MU module should be reconfigured for RPC/IPC communication
5. RPC/IPC communication can proceed normally

10.4 Flags/Status

10.4.1 SCFW to Client Flags/Status

SCFW makes use of the MU general-purpose flags to provide the client status and information. Currently these flags are only used to convey the instance (Core ID) of Cortex-M4 subsystems.

MU Fx	Use
0-2	Conveys subsystem instance (only Cortex-M4 subsystems)

10.5 Porting

The SCFW porting kit contains ports of the SCFW client API for ATF, FreeRTOS, Linux, QNX, and u-boot. All implement the same API and protocol. The variations involve license, directory structure, file names, and include paths.

```
scfw_export_atf.tar.gz
scfw_export_freertos.tar.gz
scfw_export_linux.tar.gz
scfw_export_qnx.tar.gz
scfw_export_uboot.tar.gz
scfw_export_headers.tar.gz
```

Integrating the API involves implementing the IPC layer. This primarily involves implementing:

```
void sc_call_rpc(sc_ipc_t ipc, sc_rpc_msg_t *msg, sc_bool_t no_resp)
```

This function should send the message pointed to by msg via an MU (identified by ipc), word by word. If no_resp is SC_FALSE then it should wait on a response and return it in the same structure pointed to by msg. sc_call_rpc() must be atomic and often requires a semaphore to insure this is the case. The size of the message is in the message itself. The IPC implementation can be interrupt or poll driven, directly to the MU or via an MU driver.

The IPC layer typically implements functions like:

```
sc_err_t sc_ipc_open(sc_ipc_t *ipc, sc_ipc_id_t id)
void sc_ipc_close(sc_ipc_t ipc)
void sc_ipc_read(sc_ipc_t ipc, void *data)
void sc_ipc_write(sc_ipc_t ipc, const void *data)
```

but that is up to the implementer. All that the SCFW API requires is that sc_call_rpc() be implemented.

10.6 API Message Formats

Below is a list of all API calls and their message formats to send a request and receive a response. A select few API calls do not return a response (self reset and reboot functions).

Messages are expected to be in little-endian format. The description accounts for this in the location of the parameter but within a parameter bytes may be in a reverse order depending on how the protocol is examined.

10.6.1 sc_pm_set_sys_power_mode()

Send

w[0]	func=19U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mode	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.2 sc_pm_set_partition_power_mode()

Send

w[0]	func=1U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		mode		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.3 sc_pm_get_sys_power_mode()

Send

w[0]	func=2U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mode	

10.6.4 sc_pm_partition_wake()

Send

w[0]	func=28U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.5 sc_pm_set_resource_power_mode()

Send

w[0]	func=3U		svc=2U		size=2U		ver=1U	
w[1]	undefined		mode		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.6 sc_pm_set_resource_power_mode_all()

Send

w[0]	func=22U		svc=2U		size=2U		ver=1U	
w[1]	mode		pt		exclude			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.7 sc_pm_get_resource_power_mode()

Send

w[0]	func=4U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		resource			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mode	

10.6.8 sc_pm_req_low_power_mode()

Send

w[0]	func=16U		svc=2U		size=2U		ver=1U	
w[1]	undefined		mode		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.9 sc_pm_req_cpu_low_power_mode()

Send

w[0]	func=20U		svc=2U		size=2U		ver=1U	
w[1]	wake_src		mode		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.10 sc_pm_set_cpu_resume_addr()

Send

w[0]	func=17U		svc=2U		size=4U		ver=1U	
w[1]	address (MSW)							
w[2]	address (LSW)							
w[3]	undefined		undefined		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.11 sc_pm_set_cpu_resume()

Send

w[0]	func=21U		svc=2U		size=4U		ver=1U	
w[1]	address (MSW)							
w[2]	address (LSW)							
w[3]	undefined		isPrimary		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.12 sc_pm_req_sys_if_power_mode()

Send

w[0]	func=18U		svc=2U		size=3U		ver=1U	
w[1]	hpm		sys_if		resource			
w[2]	undefined		undefined		undefined		lpm	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.13 sc_pm_set_clock_rate()

Send

w[0]	func=5U		svc=2U		size=3U		ver=1U	
------	---------	--	--------	--	---------	--	--------	--

w[1]	rate			
w[2]	undefined	clk	resource	

Receive

w[0]	err	svc=1U	size=2U	ver=1U	
w[1]	rate				

10.6.14 sc_pm_get_clock_rate()

Send

w[0]	func=6U		svc=2U		size=2U		ver=1U	
w[1]	undefined		clk		resource			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	rate							

10.6.15 sc_pm_clock_enable()

Send

w[0]	func=7U	svc=2U	size=3U	ver=1U
w[1]	enable	clk	resource	
w[2]	undefined	undefined	undefined	autog

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.16 sc_pm_set_clock_parent()

Send

w[0]	func=14U	svc=2U	size=2U	ver=1U
w[1]	parent	clk	resource	

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.17 sc_pm_get_clock_parent()

Send

w[0]	func=15U		svc=2U		size=2U		ver=1U	
w[1]	undefined		clk		resource			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		parent	

10.6.18 sc_pm_reset()

Send

w[0]	func=13U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		type	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.19 sc_pm_reset_reason()

Send

w[0]	func=10U		svc=2U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		reason	

10.6.20 sc_pm_get_reset_part()

Send

w[0]	func=26U		svc=2U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

10.6.21 sc_pm_boot()

Send

w[0]	func=8U		svc=2U		size=5U		ver=1U	
w[1]	boot_addr (MSW)							
w[2]	boot_addr (LSW)							
w[3]	resource_mu				resource_cpu			
w[4]	undefined		pt		resource_dev			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.22 sc_pm_set_boot_parm()

Send

w[0]	func=27U		svc=2U		size=5U		ver=1U	
w[1]	boot_addr (MSW)							
w[2]	boot_addr (LSW)							
w[3]	resource_mu				resource_cpu			
w[4]	undefined		undefined		resource_dev			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.23 sc_pm_reboot()

Send

w[0]	func=9U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		type	

10.6.24 sc_pm_reboot_partition()

Send

w[0]	func=12U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		type		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.25 sc_pm_reboot_continue()

Send

w[0]	func=25U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.26 sc_pm_cpu_start()

Send

w[0]	func=11U		svc=2U		size=4U		ver=1U	
w[1]	address (MSW)							
w[2]	address (LSW)							
w[3]	undefined		enable		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.27 sc_pm_cpu_reset()

Send

w[0]	func=23U		svc=2U		size=4U		ver=1U	
w[1]	address (MSW)							
w[2]	address (LSW)							
w[3]	undefined		undefined		resource			

10.6.28 sc_pm_resource_reset()

Send

w[0]	func=29U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.29 sc_pm_is_partition_started()

Send

w[0]	func=24U		svc=2U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

Receive

w[0]	result		svc=1U		size=1U		ver=1U	
------	--------	--	--------	--	---------	--	--------	--

10.6.30 sc_rm_partition_alloc()

Send

w[0]	func=1U		svc=3U		size=3U		ver=1U	
w[1]	grant		restricted		isolated		secure	
w[2]	undefined		undefined		undefined		coherent	

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

10.6.31 sc_rm_set_confidential()

Send

w[0]	func=31U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		retro		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.32 sc_rm_partition_free()

Send

w[0]	func=2U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.33 sc_rm_get_did()

Send

w[0]	func=26U		svc=3U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	result		svc=1U		size=1U		ver=1U	
------	--------	--	--------	--	---------	--	--------	--

10.6.34 sc_rm_partition_static()

Send

w[0]	func=3U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		did		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.35 sc_rm_partition_lock()

Send

w[0]	func=4U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.36 sc_rm_get_partition()

Send

w[0]	func=5U		svc=3U		size=1U		ver=1U	
------	---------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

10.6.37 sc_rm_set_parent()

Send

w[0]	func=6U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		pt_parent		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.38 sc_rm_move_all()

Send

w[0]	func=7U		svc=3U		size=2U		ver=1U	
w[1]	move_pads		move_rsrc		pt_dst		pt_src	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.39 sc_rm_assign_resource()

Send

w[0]	func=8U		svc=3U		size=2U		ver=1U	
w[1]	undefined		pt		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.40 sc_rm_set_resource_movable()

Send

w[0]	func=9U		svc=3U		size=3U		ver=1U	
w[1]	resource_lst				resource_fst			
w[2]	undefined		undefined		undefined		movable	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.41 sc_rm_set_subsys_rsrc_movable()

Send

w[0]	func=28U		svc=3U		size=2U		ver=1U	
w[1]	undefined		movable		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.42 sc_rm_set_master_attributes()

Send

w[0]	func=10U		svc=3U		size=3U		ver=1U	
w[1]	pa		sa		resource			
w[2]	undefined		undefined		undefined		smmu_bypass	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.43 sc_rm_set_master_sid()

Send

w[0]	func=11U		svc=3U		size=2U		ver=1U	
w[1]	sid		resource					

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.44 sc_rm_set_peripheral_permissions()

Send

w[0]	func=12U		svc=3U		size=2U		ver=1U	
w[1]	perm		pt		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.45 sc_rm_is_resource_owned()

Send

w[0]	func=13U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		resource			

Receive

w[0]	result		svc=1U		size=1U		ver=1U	
------	--------	--	--------	--	---------	--	--------	--

10.6.46 sc_rm_get_resource_owner()

Send

w[0]	func=33U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		resource			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

10.6.47 sc_rm_is_resource_master()

Send

w[0]	func=14U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		resource			

Receive

w[0]	result		svc=1U		size=1U		ver=1U	
------	--------	--	--------	--	---------	--	--------	--

10.6.48 sc_rm_is_resource_peripheral()

Send

w[0]	func=15U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		resource			

Receive

w[0]	result		svc=1U		size=1U		ver=1U	
------	--------	--	--------	--	---------	--	--------	--

10.6.49 sc_rm_get_resource_info()

Send

w[0]	func=16U	svc=3U	size=2U	ver=1U	
w[1]	undefined	undefined	resource		

Receive

w[0]	err	svc=1U	size=2U	ver=1U	
w[1]	undefined	undefined	sid		

10.6.50 sc_rm_memreg_alloc()

Send

w[0]	func=17U	svc=3U	size=5U	ver=1U	
w[1]	addr_start (MSW)				
w[2]	addr_start (LSW)				
w[3]	addr_end (MSW)				
w[4]	addr_end (LSW)				

Receive

w[0]	err	svc=1U	size=2U	ver=1U	
w[1]	undefined	undefined	undefined	mr	

10.6.51 sc_rm_memreg_split()

Send

w[0]	func=29U	svc=3U	size=6U	ver=1U	
w[1]	addr_start (MSW)				
w[2]	addr_start (LSW)				
w[3]	addr_end (MSW)				
w[4]	addr_end (LSW)				
w[5]	undefined	undefined	undefined	mr	

Receive

w[0]	err	svc=1U	size=2U	ver=1U	
w[1]	undefined	undefined	undefined	mr_ret	

10.6.52 sc_rm_memreg_frag()

Send

w[0]	func=32U		svc=3U		size=5U		ver=1U	
w[1]	addr_start (MSW)							
w[2]	addr_start (LSW)							
w[3]	addr_end (MSW)							
w[4]	addr_end (LSW)							

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mr_ret	

10.6.53 sc_rm_memreg_free()

Send

w[0]	func=18U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mr	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.54 sc_rm_find_memreg()

Send

w[0]	func=30U		svc=3U		size=5U		ver=1U	
w[1]	addr_start (MSW)							
w[2]	addr_start (LSW)							
w[3]	addr_end (MSW)							
w[4]	addr_end (LSW)							

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mr	

10.6.55 sc_rm_assign_memreg()

Send

w[0]	func=19U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		mr		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.56 sc_rm_set_memreg_permissions()

Send

w[0]	func=20U		svc=3U		size=2U		ver=1U	
w[1]	undefined		perm		pt		mr	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.57 sc_rm_set_memreg_iee()

Send

w[0]	func=34U		svc=3U		size=2U		ver=1U	
w[1]	undefined		rmsg		det		mr	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.58 sc_rm_is_memreg_owned()

Send

w[0]	func=21U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mr	

Receive

w[0]	result		svc=1U		size=1U		ver=1U	
------	--------	--	--------	--	---------	--	--------	--

10.6.59 sc_rm_get_memreg_info()

Send

w[0]	func=22U		svc=3U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mr	

Receive

w[0]	err		svc=1U		size=5U		ver=1U	
w[1]	addr_start (MSW)							
w[2]	addr_start (LSW)							
w[3]	addr_end (MSW)							
w[4]	addr_end (LSW)							

10.6.60 sc_rm_assign_pad()

Send

w[0]	func=23U		svc=3U		size=2U		ver=1U	
w[1]	undefined		pt		pad			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.61 sc_rm_set_pad_movable()

Send

w[0]	func=24U		svc=3U		size=3U		ver=1U	
w[1]	pad_lst				pad_fst			
w[2]	undefined		undefined		undefined		movable	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.62 sc_rm_is_pad_owned()

Send

w[0]	func=25U		svc=3U		size=2U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

w[1]	undefined	undefined	pad
------	-----------	-----------	-----

Receive

w[0]	result	svc=1U	size=1U	ver=1U
------	--------	--------	---------	--------

10.6.63 sc_rm_dump()

Send

w[0]	func=27U	svc=3U	size=1U	ver=1U
------	----------	--------	---------	--------

Receive

w[0]	undefined	svc=1U	size=1U	ver=1U
------	-----------	--------	---------	--------

10.6.64 sc_timer_set_wdog_timeout()

Send

w[0]	func=1U	svc=5U	size=2U	ver=1U
w[1]	timeout			

Receive

w[0]	err	svc=1U	size=1U	ver=1U
------	-----	--------	---------	--------

10.6.65 sc_timer_set_wdog_pre_timeout()

Send

w[0]	func=12U	svc=5U	size=2U	ver=1U
w[1]	pre_timeout			

Receive

w[0]	err	svc=1U	size=1U	ver=1U
------	-----	--------	---------	--------

10.6.66 sc_timer_set_wdog_window()

Send

w[0]	func=19U	svc=5U	size=2U	ver=1U
w[1]	window			

Receive

w[0]	err	svc=1U	size=1U	ver=1U
------	-----	--------	---------	--------

10.6.67 sc_timer_start_wdog()

Send

w[0]	func=2U		svc=5U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		lock	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.68 sc_timer_stop_wdog()

Send

w[0]	func=3U		svc=5U		size=1U		ver=1U	
------	---------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.69 sc_timer_ping_wdog()

Send

w[0]	func=4U		svc=5U		size=1U		ver=1U	
------	---------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.70 sc_timer_get_wdog_status()

Send

w[0]	func=5U		svc=5U		size=1U		ver=1U	
------	---------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=4U		ver=1U	
w[1]	timeout							
w[2]	max_timeout							
w[3]	remaining_time							

10.6.71 sc_timer_pt_get_wdog_status()

Send

w[0]	func=13U		svc=5U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		pt	

Receive

w[0]	err		svc=1U		size=4U		ver=1U	
w[1]	timeout							
w[2]	remaining_time							
w[3]	undefined		undefined		undefined		enb	

10.6.72 sc_timer_set_wdog_action()

Send

w[0]	func=10U		svc=5U		size=2U		ver=1U	
w[1]	undefined		undefined		action		pt	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.73 sc_timer_set_rtc_time()

Send

w[0]	func=6U		svc=5U		size=3U		ver=1U	
w[1]	day		mon		year			
w[2]	undefined		sec		min		hour	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.74 sc_timer_get_rtc_time()

Send

w[0]	func=7U		svc=5U		size=1U		ver=1U	
------	---------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=3U		ver=1U	
w[1]	day		mon		year			
w[2]	undefined		sec		min		hour	

10.6.75 sc_timer_get_rtc_sec1970()

Send

w[0]	func=9U		svc=5U		size=1U		ver=1U	
------	---------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	sec							

10.6.76 sc_timer_set_rtc_alarm()

Send

w[0]	func=8U		svc=5U		size=3U		ver=1U	
w[1]	day		mon		year			
w[2]	undefined		sec		min		hour	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.77 sc_timer_set_rtc_periodic_alarm()

Send

w[0]	func=14U		svc=5U		size=2U		ver=1U	
w[1]	sec							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.78 sc_timer_cancel_rtc_alarm()

Send

w[0]	func=15U		svc=5U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.79 sc_timer_set_rtc_calb()

Send

w[0]	func=11U		svc=5U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		count	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.80 sc_timer_set_sysctr_alarm()

Send

w[0]	func=16U		svc=5U		size=3U		ver=1U	
w[1]	ticks (MSW)							
w[2]	ticks (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.81 sc_timer_set_sysctr_periodic_alarm()

Send

w[0]	func=17U		svc=5U		size=3U		ver=1U	
w[1]	ticks (MSW)							
w[2]	ticks (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.82 sc_timer_cancel_sysctr_alarm()

Send

w[0]	func=18U		svc=5U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.83 sc_pad_set_mux()

Send

w[0]	func=1U		svc=6U		size=3U		ver=1U	
w[1]	config		mux		pad			
w[2]	undefined		undefined		undefined		iso	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.84 sc_pad_get_mux()

Send

w[0]	func=6U		svc=6U		size=2U		ver=1U	
w[1]	undefined		undefined		pad			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		iso		config		mux	

10.6.85 sc_pad_set_gp()

Send

w[0]	func=2U		svc=6U		size=3U		ver=1U	
w[1]	ctrl							
w[2]	undefined		undefined		pad			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.86 sc_pad_get_gp()

Send

w[0]	func=7U	svc=6U	size=2U	ver=1U	
w[1]	undefined	undefined	pad		

Receive

w[0]	err	svc=1U	size=2U	ver=1U	
w[1]	ctrl				

10.6.87 sc_pad_set_wakeup()

Send

w[0]	func=4U	svc=6U	size=2U	ver=1U	
w[1]	undefined	wakeup	pad		

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.88 sc_pad_get_wakeup()

Send

w[0]	func=9U	svc=6U	size=2U	ver=1U	
w[1]	undefined	undefined	pad		

Receive

w[0]	err	svc=1U	size=2U	ver=1U	
w[1]	undefined	undefined	undefined	wakeup	

10.6.89 sc_pad_set_all()

Send

w[0]	func=5U	svc=6U	size=4U	ver=1U	
w[1]	ctrl				
w[2]	config	mux	pad		
w[3]	undefined	undefined	wakeup	iso	

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.90 sc_pad_get_all()

Send

w[0]	func=10U		svc=6U		size=2U		ver=1U	
w[1]	undefined		undefined		pad			

Receive

w[0]	err		svc=1U		size=3U		ver=1U	
w[1]	ctrl							
w[2]	wakeup		iso		config		mux	

10.6.91 sc_pad_set()

Send

w[0]	func=15U		svc=6U		size=3U		ver=1U	
w[1]	val							
w[2]	undefined		undefined		pad			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.92 sc_pad_get()

Send

w[0]	func=16U		svc=6U		size=2U		ver=1U	
w[1]	undefined		undefined		pad			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	val							

10.6.93 sc_pad_config()

Send

w[0]	func=17U		svc=6U		size=3U		ver=1U	
w[1]	val							

w[2]	undefined	undefined	pad
------	-----------	-----------	-----

Receive

w[0]	err	svc=1U	size=1U	ver=1U
------	-----	--------	---------	--------

10.6.94 sc_pad_set_gp_28fdsoi()

Send

w[0]	func=11U	svc=6U	size=2U	ver=1U
w[1]	ps	dse	pad	

Receive

w[0]	err	svc=1U	size=1U	ver=1U
------	-----	--------	---------	--------

10.6.95 sc_pad_get_gp_28fdsoi()

Send

w[0]	func=12U	svc=6U	size=2U	ver=1U
w[1]	undefined	undefined	pad	

Receive

w[0]	err	svc=1U	size=2U	ver=1U
w[1]	undefined	undefined	ps	dse

10.6.96 sc_pad_set_gp_28fdsoi_hsic()

Send

w[0]	func=3U	svc=6U	size=3U	ver=1U
w[1]	pus	dse	pad	
w[2]	undefined	pue	pke	hys

Receive

w[0]	err	svc=1U	size=1U	ver=1U
------	-----	--------	---------	--------

10.6.97 sc_pad_get_gp_28fdsoi_hsic()

Send

w[0]	func=8U	svc=6U	size=2U	ver=1U	
w[1]	undefined	undefined	pad		

Receive

w[0]	err	svc=1U	size=3U	ver=1U	
w[1]	pke	hys	pus	dse	
w[2]	undefined	undefined	undefined	pue	

10.6.98 sc_pad_set_gp_28fdsoi_comp()

Send

w[0]	func=13U	svc=6U	size=3U	ver=1U	
w[1]	rasrcp	compen	pad		
w[2]	psw_ovr	nasrc_sel	fastfrz	rasrcn	

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.99 sc_pad_get_gp_28fdsoi_comp()

Send

w[0]	func=14U	svc=6U	size=2U	ver=1U	
w[1]	undefined	undefined	pad		

Receive

w[0]	err	svc=1U	size=3U	ver=1U	
w[1]	nasrc	rasrcn	rasrcp	compen	
w[2]	psw_ovr	compok	nasrc_sel	fastfrz	

10.6.100 sc_misc_set_control()

Send

w[0]	func=1U	svc=7U	size=4U	ver=1U	
------	---------	--------	---------	--------	--

w[1]	ctrl			
w[2]	val			
w[3]	undefined	undefined	resource	

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.101 sc_misc_get_control()

Send

w[0]	func=2U	svc=7U	size=3U	ver=1U	
w[1]	ctrl				
w[2]	undefined	undefined	resource		

Receive

w[0]	err	svc=1U	size=2U	ver=1U	
w[1]	val				

10.6.102 sc_misc_set_max_dma_group()

Send

w[0]	func=4U	svc=7U	size=2U	ver=1U	
w[1]	undefined	undefined	max	pt	

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.103 sc_misc_set_dma_group()

Send

w[0]	func=5U	svc=7U	size=2U	ver=1U
w[1]	undefined	group	resource	

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.104 sc_misc_debug_out()

Send

w[0]	func=10U		svc=7U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		ch	

Receive

w[0]	undefined		svc=1U		size=1U		ver=1U	
------	-----------	--	--------	--	---------	--	--------	--

10.6.105 sc_misc_waveform_capture()

Send

w[0]	func=6U		svc=7U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		enable	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.106 sc_misc_build_info()

Send

w[0]	func=15U		svc=7U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	undefined		svc=1U		size=3U		ver=1U	
w[1]	build							
w[2]	commit							

10.6.107 sc_misc_api_ver()

Send

w[0]	func=35U		svc=7U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	undefined		svc=1U		size=3U		ver=1U	
w[1]	cl_min				cl_maj			
w[2]	sv_min				sv_maj			

10.6.108 sc_misc_unique_id()

Send

w[0]	func=19U		svc=7U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	undefined		svc=1U		size=3U		ver=1U	
w[1]	id_l							
w[2]	id_h							

10.6.109 sc_misc_set_ari()

Send

w[0]	func=3U		svc=7U		size=3U		ver=1U	
w[1]	resource_mst				resource			
w[2]	undefined		enable		ari			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.110 sc_misc_boot_status()

Send

w[0]	func=7U		svc=7U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		status	

10.6.111 sc_misc_boot_done()

Send

w[0]	func=14U		svc=7U		size=2U		ver=1U	
w[1]	undefined		undefined		cpu			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.112 sc_misc_otf_fuse_read()

Send

w[0]	func=11U		svc=7U		size=2U		ver=1U	
w[1]	word							

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	val							

10.6.113 sc_misc_otf_fuse_write()

Send

w[0]	func=17U		svc=7U		size=3U		ver=1U	
w[1]	word							
w[2]	val							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.114 sc_misc_set_temp()

Send

w[0]	func=12U		svc=7U		size=3U		ver=1U	
w[1]	celsius				resource			
w[2]	undefined		undefined		tenths		temp	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.115 sc_misc_get_temp()

Send

w[0]	func=13U		svc=7U		size=2U		ver=1U	
w[1]	undefined		temp		resource			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		tenths		celsius			

10.6.116 sc_misc_get_boot_dev()

Send

w[0]	func=16U		svc=7U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	undefined		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined				dev	

10.6.117 sc_misc_get_boot_type()

Send

w[0]	func=33U		svc=7U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		type	

10.6.118 sc_misc_get_boot_container()

Send

w[0]	func=36U		svc=7U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		idx	

10.6.119 sc_misc_get_button_status()

Send

w[0]	func=18U		svc=7U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	undefined		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		status	

10.6.120 sc_misc_rompatch_checksum()

Send

w[0]	func=26U		svc=7U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	checksum							

10.6.121 sc_misc_board_ioctl()

Send

w[0]	func=34U		svc=7U		size=4U		ver=1U	
w[1]	parm1							
w[2]	parm2							
w[3]	parm3							

Receive

w[0]	err		svc=1U		size=4U		ver=1U	
w[1]	parm1							
w[2]	parm2							
w[3]	parm3							

10.6.122 sc_seco_image_load()

Send

w[0]	func=1U		svc=9U		size=7U		ver=1U	
w[1]	addr_src (MSW)							
w[2]	addr_src (LSW)							
w[3]	addr_dst (MSW)							
w[4]	addr_dst (LSW)							
w[5]	len							
w[6]	undefined		undefined		undefined		fw	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.123 sc_seco_authenticate()

Send

w[0]	func=2U		svc=9U		size=4U		ver=1U	
w[1]	addr (MSW)							
w[2]	addr (LSW)							
w[3]	undefined		undefined		undefined		cmd	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.124 sc_seco_enh_authenticate()

Send

w[0]	func=24U		svc=9U		size=6U		ver=1U	
w[1]	addr (MSW)							
w[2]	addr (LSW)							
w[3]	mask1							
w[4]	mask2							
w[5]	undefined		undefined		undefined		cmd	

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.125 sc_seco_forward_lifecycle()

Send

w[0]	func=3U		svc=9U		size=2U		ver=1U	
w[1]	change							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.126 sc_seco_return_lifecycle()

Send

w[0]	func=4U		svc=9U		size=3U		ver=1U	
w[1]	addr (MSW)							
w[2]	addr (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.127 sc_seco_commit()

Send

w[0]	func=5U		svc=9U		size=2U		ver=1U	
w[1]	info							

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	info							

10.6.128 sc_seco_attest_mode()

Send

w[0]	func=6U		svc=9U		size=2U		ver=1U	
w[1]	mode							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.129 sc_seco_attest()

Send

w[0]	func=7U		svc=9U		size=3U		ver=1U	
w[1]	nonce (MSW)							
w[2]	nonce (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.130 sc_seco_get_attest_pkey()

Send

w[0]	func=8U		svc=9U		size=3U		ver=1U	
w[1]	addr (MSW)							
w[2]	addr (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.131 sc_seco_get_attest_sign()

Send

w[0]	func=9U		svc=9U		size=3U		ver=1U	
w[1]	addr (MSW)							
w[2]	addr (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.132 sc_seco_attest_verify()

Send

w[0]	func=10U		svc=9U		size=3U		ver=1U	
w[1]	addr (MSW)							
w[2]	addr (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.133 sc_seco_gen_key_blob()

Send

w[0]	func=11U		svc=9U		size=7U		ver=1U	
w[1]	load_addr (MSW)							
w[2]	load_addr (LSW)							

w[3]	export_addr (MSW)			
w[4]	export_addr (LSW)			
w[5]	id			
w[6]	undefined	undefined	max_size	

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.134 sc_seco_load_key()

Send

w[0]	func=12U	svc=9U	size=4U	ver=1U	
w[1]	addr (MSW)				
w[2]	addr (LSW)				
w[3]	id				

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.135 sc_seco_get_mp_key()

Send

w[0]	func=13U	svc=9U	size=4U	ver=1U	
w[1]	dst_addr (MSW)				
w[2]	dst_addr (LSW)				
w[3]	undefined	undefined	dst_size		

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.136 sc_seco_update_mpmr()

Send

w[0]	func=14U	svc=9U	size=4U	ver=1U	
w[1]	addr (MSW)				
w[2]	addr (LSW)				

w[3]	undefined		undefined		lock		size	
------	-----------	--	-----------	--	------	--	------	--

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.137 sc_seco_get_mp_sign()

Send

w[0]	func=15U		svc=9U		size=6U		ver=1U	
w[1]	msg_addr (MSW)							
w[2]	msg_addr (LSW)							
w[3]	dst_addr (MSW)							
w[4]	dst_addr (LSW)							
w[5]	dst_size				msg_size			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.138 sc_seco_build_info()

Send

w[0]	func=16U		svc=9U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	undefined		svc=1U		size=3U		ver=1U	
w[1]	version							
w[2]	commit							

10.6.139 sc_seco_chip_info()

Send

w[0]	func=17U		svc=9U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=4U		ver=1U	
w[1]	uid_l							

w[2]	uid_h	
w[3]	monotonic	lc

10.6.140 sc_seco_enable_debug()

Send

w[0]	func=18U	svc=9U	size=3U	ver=1U	
w[1]	addr (MSW)				
w[2]	addr (LSW)				

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.141 sc_seco_get_event()

Send

w[0]	func=19U	svc=9U	size=2U	ver=1U	
w[1]	undefined	undefined	undefined	idx	

Receive

w[0]	err	svc=1U	size=2U	ver=1U	
w[1]	event				

10.6.142 sc_seco_fuse_write()

Send

w[0]	func=20U	svc=9U	size=3U	ver=1U	
w[1]	addr (MSW)				
w[2]	addr (LSW)				

Receive

w[0]	err	svc=1U	size=1U	ver=1U	
------	-----	--------	---------	--------	--

10.6.143 sc_seco_patch()

Send

w[0]	func=21U		svc=9U		size=3U		ver=1U	
w[1]	addr (MSW)							
w[2]	addr (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.144 sc_seco_set_mono_counter_partition()

Send

w[0]	func=28U		svc=9U		size=2U		ver=1U	
w[1]	undefined		undefined		she			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	undefined		undefined		she			

10.6.145 sc_seco_set_fips_mode()

Send

w[0]	func=29U		svc=9U		size=2U		ver=1U	
w[1]	undefined		undefined		undefined		mode	

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	reason							

10.6.146 sc_seco_start_rng()

Send

w[0]	func=22U		svc=9U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	status							

10.6.147 sc_seco_sab_msg()

Send

w[0]	func=23U		svc=9U		size=3U		ver=1U	
w[1]	addr (MSW)							
w[2]	addr (LSW)							

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.148 sc_seco_secvio_enable()

Send

w[0]	func=25U		svc=9U		size=1U		ver=1U	
------	----------	--	--------	--	---------	--	--------	--

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.149 sc_seco_secvio_config()

Send

w[0]	func=26U		svc=9U		size=7U		ver=1U	
w[1]	data0							
w[2]	data1							
w[3]	data2							
w[4]	data3							
w[5]	data4							
w[6]	undefined		size		access		id	

Receive

w[0]	err		svc=1U		size=6U		ver=1U	
w[1]	data0							
w[2]	data1							
w[3]	data2							
w[4]	data3							
w[5]	data4							

10.6.150 sc_seco_secvio_dgo_config()

Send

w[0]	func=27U		svc=9U		size=3U		ver=1U	
w[1]	data							
w[2]	undefined		undefined		access		id	

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	data							

10.6.151 sc_irq_enable()

Send

w[0]	func=1U		svc=8U		size=3U		ver=1U	
w[1]	mask							
w[2]	enable		group		resource			

Receive

w[0]	err		svc=1U		size=1U		ver=1U	
------	-----	--	--------	--	---------	--	--------	--

10.6.152 sc_irq_status()

Send

w[0]	func=2U		svc=8U		size=2U		ver=1U	
w[1]	undefined		group		resource			

Receive

w[0]	err		svc=1U		size=2U		ver=1U	
w[1]	status							

Chapter 11

Module Index

11.1 Modules

Here is a list of all modules:

IRQ: Interrupt Service	99
MISC: Miscellaneous Service	104
PAD: Pad Service	118
PM: Power Management Service	136
RM: Resource Management Service	160
SECO: Security Service	188
TIMER: Timer Service	214

Chapter 12

File Index

12.1 File List

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

platform/config/mx8qx/ pads.h	
Header file used to configure SoC pad list	227
platform/main/ ipc.h	
Header file for the IPC implementation	233
platform/main/ types.h	
Header file containing types used across multiple service APIs	235
platform/svc/irq/ api.h	
Header file containing the public API for the System Controller (SC) Interrupt (IRQ) function	253
platform/svc/misc/ api.h	
Header file containing the public API for the System Controller (SC) Miscellaneous (MISC) function	255
platform/svc/pad/ api.h	
Header file containing the public API for the System Controller (SC) Pad Control (PAD) function	258
platform/svc/pm/ api.h	
Header file containing the public API for the System Controller (SC) Power Management (PM) function	261
platform/svc/rm/ api.h	
Header file containing the public API for the System Controller (SC) Resource Management (RM) function	266
platform/svc/seco/ api.h	
Header file containing the public API for the System Controller (SC) Security (SECO) function	269
platform/svc/timer/ api.h	
Header file containing the public API for the System Controller (SC) Timer function	271

Chapter 13

Module Documentation

13.1 IRQ: Interrupt Service

Module for the Interrupt (IRQ) service.

Macros

- `#define SC_IRQ_NUM_GROUP 7U`
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.
- `#define SC_IRQ_GROUP_REBOOTED 5U`
Partition reboot complete.
- `#define SC_IRQ_GROUP_REBOOT 6U`
Partition reboot starting.

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.
- #define `SC_IRQ_USR1` (1U << 2U)
User defined 1.
- #define `SC_IRQ_USR2` (1U << 3U)
User defined 2.
- #define `SC_IRQ_BC_PAD` (1U << 4U)
Pad wakeup (broadcast to all partitions)
- #define `SC_IRQ_SW_WAKE` (1U << 5U)
Software requested wake.
- #define `SC_IRQ_SECVIO` (1U << 6U)
Security violation.

Defines for `sc_irq_sysctr_t`

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

13.1.1 Detailed Description

Module for the Interrupt (IRQ) service.

13.1.2 Function Documentation

13.1.2.1 `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

13.1.2.2 `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.

13.2 MISC: 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_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_bt_t`

- `#define SC_MISC_BT_PRIMARY 0U`
Primary boot.
- `#define SC_MISC_BT_SECONDARY 1U`
Secondary boot.
- `#define SC_MISC_BT_RECOVERY 2U`
Recovery boot.
- `#define SC_MISC_BT_MANUFACTURE 3U`
Manufacture boot.
- `#define SC_MISC_BT_SERIAL 4U`
Serial boot.

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.

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_api_ver` (`sc_ipc_t` ipc, `uint16_t` *cl_maj, `uint16_t` *cl_min, `uint16_t` *sv_maj, `uint16_t` *sv_min)
This function is used to return the SCFW API versions.
- `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.
- `sc_err_t sc_misc_get_boot_container` (`sc_ipc_t ipc`, `uint8_t *idx`)
This function returns the boot container index.
- `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.
- `sc_err_t sc_misc_board_ioctl` (`sc_ipc_t ipc`, `uint32_t *parm1`, `uint32_t *parm2`, `uint32_t *parm3`)
This function calls the board IOCTL function.

13.2.1 Detailed Description

Module for the Miscellaneous (MISC) service.

13.2.2 Function Documentation

13.2.2.1 `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.

13.2.2.2 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.

13.2.2.3 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*.

13.2.2.4 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\(\)](#).

13.2.2.5 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

13.2.2.6 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

13.2.2.7 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)

13.2.2.8 sc_misc_api_ver()

```
void sc_misc_api_ver (
    sc_ipc_t ipc,
    uint16_t * cl_maj,
    uint16_t * cl_min,
    uint16_t * sv_maj,
    uint16_t * sv_min )
```

This function is used to return the SCFW API versions.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>cl_maj</i>	pointer to return major part of client version
out	<i>cl_min</i>	pointer to return minor part of client version
out	<i>sv_maj</i>	pointer to return major part of SCFW version
out	<i>sv_min</i>	pointer to return minor part of SCFW version

Client version is the version of the API ported to and used by the caller. SCFW version is the version of the SCFW binary running on the CPU.

Note a major version difference indicates a break in compatibility.

13.2.2.9 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]

13.2.2.10 `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.

13.2.2.11 `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.

13.2.2.12 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

13.2.2.13 sc_misc_otp_fuse_read()

```
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.

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

13.2.2.14 sc_misc_otf_fuse_write()

```
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.

Only the owner of the SC_R_SYSTEM resource or a partition with access permissions to SC_R_SYSTEM can do this.

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 SECO API Reference Guide 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 caller does not have SC_R_SYSTEM access
- SC_ERR_NOACCESS if write operation failed
- SC_ERR_LOCKED if write operation is locked

13.2.2.15 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
- SC_ERR_NOACCESS if caller does not own the resource
- SC_ERR_NOPOWER if power domain of resource not powered

13.2.2.16 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)
- SC_ERR_NOPOWER if power domain of resource not powered

13.2.2.17 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

13.2.2.18 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

13.2.2.19 sc_misc_get_boot_container()

```
sc_err_t sc_misc_get_boot_container (
    sc_ipc_t ipc,
    uint8_t * idx )
```

This function returns the boot container index.

Parameters

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

Return *idx* = 1 for first container, 2 for second.

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors code:

- SC_ERR_UNAVAILABLE if index not passed by ROM

13.2.2.20 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

13.2.2.21 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).

13.2.2.22 sc_misc_board_ioctl()

```
sc_err_t sc_misc_board_ioctl (
    sc_ipc_t ipc,
    uint32_t * parm1,
    uint32_t * parm2,
    uint32_t * parm3 )
```

This function calls the board IOCTL function.

Parameters

in	<i>ipc</i>	IPC handle
in, out	<i>parm1</i>	pointer to pass parameter 1
in, out	<i>parm2</i>	pointer to pass parameter 2
in, out	<i>parm3</i>	pointer to pass parameter 3

Returns

Returns and error code (SC_ERR_NONE = success).

13.3 PAD: 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.
- `sc_err_t sc_pad_config` (`sc_ipc_t ipc`, `sc_pad_t pad`, `uint32_t val`)
This function writes a configuration register.

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.

13.3.1 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 SPI0_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.

```
0 /* Configure I2C_0 SCL pad */
1 sc_pad_set_mux(ipc, SC_P_MIPI_CSI0_GPIO0_00, 1, SC_PAD_CONFIG_OD_IN, SC_PAD_ISO_OFF);
2 sc_pad_set_gp_28fdsoi(ipc, SC_P_MIPI_CSI0_GPIO0_00, SC_PAD_28FDSOI_DSE_18V_1MA, SC_PAD_28FDSOI_PS_PU);
3
4 /* Configure I2C_0 SDA pad */
5 sc_pad_set_mux(ipc, SC_P_MIPI_CSI0_GPIO0_01, 1, SC_PAD_CONFIG_OD_IN, SC_PAD_ISO_OFF);
6 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.

```
7 /* Configure SPI0 SCK pad (dual-voltage) */
9 sc_pad_set_mux(ipc, SC_P_SPI0_SCK, 0, SC_PAD_CONFIG_NORMAL, SC_PAD_ISO_OFF);
10 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).

13.3.2 Typedef Documentation

13.3.2.1 `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.

13.3.2.2 `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.

13.3.2.3 `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.

13.3.2.4 `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.

13.3.2.5 `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.

13.3.3 Function Documentation

13.3.3.1 `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.

13.3.3.2 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.

13.3.3.3 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.

13.3.3.4 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.

13.3.3.5 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.

13.3.3.6 `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.

13.3.3.7 `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.

13.3.3.8 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.

13.3.3.9 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.

13.3.3.10 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.

13.3.3.11 sc_pad_config()

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

This function writes a configuration register.

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

Use to configure various HSIC and NAND congiruation settings.

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.

13.3.3.12 `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.

13.3.3.13 `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.

13.3.3.14 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.

13.3.3.15 `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.

13.3.3.16 `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).

13.3.3.17 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

Parameters

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.

13.4 PM: 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_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_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.
- `#define SC_PM_RESET_REASON_V2X_DEBUG 13U`
V2X debug switch.

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_partition_wake (sc_ipc_t ipc, sc_rm_pt_t pt)`
This function sends a wake interrupt to 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 specifies 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_get_reset_part](#) (sc_ipc_t ipc, [sc_rm_pt_t](#) *pt)
This function gets the partition that caused a reset.
- [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.
- [sc_err_t sc_pm_set_boot_parm](#) (sc_ipc_t ipc, [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 change the boot parameters for 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_reboot_continue](#) (sc_ipc_t ipc, [sc_rm_pt_t](#) pt)
This function is used to continue the 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.
- void [sc_pm_cpu_reset](#) (sc_ipc_t ipc, [sc_rsrc_t](#) resource, [sc_faddr_t](#) address)
This function is used to reset a CPU.
- [sc_err_t sc_pm_resource_reset](#) (sc_ipc_t ipc, [sc_rsrc_t](#) resource)
This function is used to reset a peripheral.
- [sc_bool_t sc_pm_is_partition_started](#) (sc_ipc_t ipc, [sc_rm_pt_t](#) pt)
This function returns a bool indicating if a partition was started.

13.4.1 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.

```
0 sc_pm_clock_rate_t rate = SC_160MHZ;
```

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

```
1 /* Powerup UART 0 */
3 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.

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

Then enable the clock.

```
7 /* Enable UART 0 clock */
9 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.

13.4.2 Typedef Documentation

13.4.2.1 `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.

13.4.3 Function Documentation

13.4.3.1 `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 or a partition with access permissions to SC_R_SYSTEM 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 does not have SC_R_SYSTEM access

See also

[sc_pm_set_sys_power_mode\(\)](#).

13.4.3.2 `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_PM_PW_MODE_OFF,
- SC_ERR_NOACCESS if caller's partition is not the owner or parent of *pt*

This function can only be used to turn off a partition by calling with mode equal to SC_PM_PW_MODE_OFF. After turning off, the partition can be booted with [sc_pm_reboot_partition\(\)](#) or [sc_pm_boot\(\)](#). It cannot be used to turn off the calling partition as the MU could not return the an error response.

For dynamic power management of a partition, use [sc_pm_req_low_power_mode\(\)](#), [sc_pm_req_cpu_low_power_mode\(\)](#), and [sc_pm_req_sys_if_power_mode\(\)](#) with a WFI for controlled power state transitions.

13.4.3.3 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

13.4.3.4 sc_pm_partition_wake()

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

This function sends a wake interrupt to a partition.

Parameters

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

Returns

Returns an error code (SC_ERR_NONE = success).

An SC_IRQ_SW_WAKE interrupt is sent to all MUs owned by the partition that have this interrupt enabled. The CPU using an MU will exit a low-power state to service the MU interrupt.

Return errors:

- SC_ERR_PARM if invalid partition

13.4.3.5 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_PARM if resource is the MU used to make the call,
- 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.

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.

Note 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. Bus 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.).

When the resource transitions to the SC_PM_PW_MODE_OFF, all of the settings, including clock rate, will be lost; immaterial of the state of other resources in the same power domain.

13.4.3.6 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 (with grant) 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.

13.4.3.7 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.

13.4.3.8 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 specifies 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_R_A53_2, SC_R_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).

13.4.3.9 `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).

13.4.3.10 `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

Note the address is limited by the hardware implementation. See the [CPU Start Address](#) section in the Porting Guide.

13.4.3.11 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

Note the address is limited by the hardware implementation. See the [CPU Start Address](#) section in the Porting Guide.

13.4.3.12 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_PARM. 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).

13.4.3.13 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

Note the actual rate is returned in *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.

13.4.3.14 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

This function returns the actual clock rate of the hardware. This rate may be different from the original requested clock rate if the resource is set to a low power mode.

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

13.4.3.15 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 (with grant) of the owner,
- SC_ERR_UNAVAILABLE if clock not applicable to this resource

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

13.4.3.16 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 (with grant) 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.

13.4.3.17 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.

13.4.3.18 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 or a partition with access permissions to SC_R_SYSTEM 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 cannot access SC_R_SYSTEM

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

13.4.3.19 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 the 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 resets may trigger a system POR and the original reason will be lost.

Returns

Returns an error code (SC_ERR_NONE = success).

13.4.3.20 sc_pm_get_reset_part()

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

This function gets the partition that caused a reset.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>pt</i>	pointer to return the resetting partition

If the reset reason obtained via [sc_pm_reset_reason\(\)](#) is POR then the result from this function will be 0. Some SECO causes of reset will also return 0.

Note depending on the connection of the WDOG_OUT signal and the OTP programming of the PMIC, some resets may trigger a system POR and the partition info will be lost.

Returns

Returns an error code (SC_ERR_NONE = success).

13.4.3.21 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

This must be used to boot a partition. Only a partition booted this way can be rebooted using the watchdog, [sc_pm_boot\(\)](#) or [sc_pm_reboot_partition\(\)](#).

Note the address is limited by the hardware implementation. See the [CPU Start Address](#) section in the Porting Guide.

13.4.3.22 `sc_pm_set_boot_parm()`

```

sc_err_t sc_pm_set_boot_parm (
    sc_ipc_t ipc,
    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 change the boot parameters for a partition.

Parameters

in	<i>ipc</i>	IPC handle
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 (0=none)
in	<i>resource_dev</i>	ID of the boot device that must be powered (0=none)

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource, or addr

This function can be used to change the boot parameters for a partition. This can be useful if a partitions reboots differently from the initial boot done via `sc_pm_boot()` or via ROM.

Note the address is limited by the hardware implementation. See the [CPU Start Address](#) section in the Porting Guide.

13.4.3.23 `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 or SC_PM_RESET_TYPE_BOARD, then does nothing.

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

13.4.3.24 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 or SC_PM_RESET_TYPE_BOARD, then returns SC_ERR_PARM as these are not supported.

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* and the caller does not have access to SC_R_SYSTEM

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.

If board_reboot_part() returns a non-0 mask, then the reboot will be delayed until all partitions indicated in the mask have called [sc_pm_reboot_continue\(\)](#) to continue the boot.

13.4.3.25 sc_pm_reboot_continue()

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

This function is used to continue the reboot a partition.

Parameters

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

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid partition

13.4.3.26 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

This function is usually used to start a secondary CPU in the same partition as the caller. It is not used to start the first CPU in a dedicated partition. That would be started by calling [sc_pm_boot\(\)](#).

A CPU started with [sc_pm_cpu_start\(\)](#) will not restart as a result of a watchdog event or calling [sc_pm_reboot\(\)](#) or [sc_pm_reboot_partition\(\)](#). Those will reboot that partition which will start the CPU started with [sc_pm_boot\(\)](#).

Note the address is limited by the hardware implementation. See the [CPU Start Address](#) section in the Porting Guide.

13.4.3.27 sc_pm_cpu_reset()

```
void sc_pm_cpu_reset (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_faddr_t address )
```

This function is used to reset a CPU.

Parameters

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

This function does not return anything as the calling core may have been reset. It can still fail if the resource or address is invalid. It can also fail if the caller's partition is not the owner of the CPU, not the parent of the CPU resource owner, or has access to SC_R_SYSTEM. Will also fail if the resource is not powered on. No indication of failure is returned.

Note this just resets the CPU. None of the peripherals or bus fabric used by the CPU is reset. State configured in the SCFW is not reset. The SW running on the core has to understand and deal with this.

The address is limited by the hardware implementation. See the [CPU Start Address](#) section in the Porting Guide.

13.4.3.28 sc_pm_resource_reset()

```
sc_err_t sc_pm_resource_reset (
    sc_ipc_t ipc,
    sc_rsrc_t resource )
```

This function is used to reset a peripheral.

Parameters

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

This function will reset a resource. Most resources cannot be reset unless the SoC design specifically allows it. In the case on MUs, the IPC/RPC protocol is also reset. Note a caller cannot reset an MU that this API call is sent on.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid resource,
- SC_ERR_PARM if resource is the MU used to make the call,

- SC_ERR_NOACCESS if caller's partition is not the resource owner or parent (with grant) of the owner,
- SC_ERR_BUSY if the resource cannot be reset due to power state of buses,
- SC_ERR_UNAVAILABLE if the resource cannot be reset due to hardware limitations

13.4.3.29 sc_pm_is_partition_started()

```
sc_bool_t sc_pm_is_partition_started (
    sc_ipc_t ipc,
    sc_rm_pt_t pt )
```

This function returns a bool indicating if a partition was started.

Parameters

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

Returns

Returns a bool (SC_TRUE = started).

Note this indicates if a partition was started. It does not indicate if a partition is currently running or in a low power state.

13.5 RM: 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.
- typedef [uint8_t](#) [sc_rm_det_t](#)
This type is used to indicate memory region transactions should detour to the IEE.
- typedef [uint8_t](#) [sc_rm_rmsg_t](#)
This type is used to assign an RMSG value to a memory region.

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](#).
- #define [SC_RM_DET_W](#) 1U
Width of [sc_rm_det_t](#).
- #define [SC_RM_RMSG_W](#) 4U
Width of [sc_rm_rmsg_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_err_t sc_rm_get_resource_owner` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_rm_pt_t` *pt)

This function is used to get the owner 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 an existing memory region.

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

This function requests that the SC fragment 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_err_t sc_rm_set_memreg_iee` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`, `sc_rm_det_t det`, `sc_rm_rmsg_t rmsg`)
This function configures the IEE parameters 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.

13.5.1 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, pt, pt_m4_0, SC_TRUE, SC_TRUE);
32
33 /* Allow all to access the SEMA42 */
34 sc_rm_set_peripheral_permissions(ipc, 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.

```

0  sc_rm_pt_t pt_m4_0;
1  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.

```

2  //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()`.

```

4  /* Mark all resources as not movable */
5  sc_rm_set_resource_movable(ipc, SC_R_ALL, SC_R_ALL, SC_FALSE);
6  sc_rm_set_resource_movable(ipc, SC_P_ALL, SC_P_ALL, SC_FALSE);
7

```

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.

```

8  /* Allocate M4_0 partition */
10 sc_rm_partition_alloc(ipc, &pt_m4_0, SC_FALSE, SC_TRUE, SC_FALSE, SC_TRUE,
11     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.

```
12 /* Mark all M4_0 subsystem resources as movable */
14 sc_rm_set_subsys_rsrc_movable(ipc, SC_R_M4_0_PID0, SC_TRUE);
15 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.

```
16 /* Keep some resources in the parent partition */
18 sc_rm_set_resource_movable(ipc, SC_R_M4_0_PID1, SC_R_M4_0_PID4,
19     SC_FALSE);
20 sc_rm_set_resource_movable(ipc, SC_R_M4_0_MU_0A0, SC_R_M4_0_MU_0A3,
21     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.

```
22 /* Move some resource not in the M4_0 subsystem */
24 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.

```
25 /* Move everything flagged as movable */
30 sc_rm_move_all(ipc, pt, 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.

```
31 /* Allow all to access the SEMA42 */
33 sc_rm_set_peripheral_permissions(ipc, pt_m4_0, SC_R_M4_0_SEMA42,
34     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.

```
35 /* Move M4_0 TCM */
37 sc_rm_find_memreg(ipc, &mr_m4_0, 0x034FE0000, 0x034FE0000);
38 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.

```
39 /* Split DDR space, assign 0x88000000-0x8FFFFFFF to CM4 */
41 sc_rm_find_memreg(ipc, &mr_ddr1, 0x080000000, 0x080000000);
42 sc_rm_memreg_split(ipc, mr_ddr1, &mr_ddr2, 0x090000000, 0xFFFFFFFF);
43
44 /* Reserve DDR for M4_0 */
45 sc_rm_memreg_split(ipc, mr_ddr1, &mr_m4_0, 0x088000000, 0x08FFFFFFF);
46 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.

```
47 //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.

13.5.2 Typedef Documentation

13.5.2.1 `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.

13.5.2.2 `sc_rm_rmsg_t`

```
typedef uint8_t sc_rm_rmsg_t
```

This type is used to assign an RMSG value to a memory region.

This value is sent to the IEE.

13.5.3 Function Documentation

13.5.3.1 `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. Basically, if TrustZone SW is used, the Cortex-A cores and peripherals the TZ SW will use should be in a secure partition. Almost all other partitions (for a non-secure OS or M4 cores) should be in non-secure partitions.

Isolated should be true for almost all partitions. The exception is the non-secure partition for a Cortex-A core used to run a non-secure OS. This isn't isolated by domain but is instead isolated by the TZ security hardware.

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. This is only used when creating a sub-partition with no CPU. It's useful to separate out a master and the memory it uses.

13.5.3.2 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.

13.5.3.3 `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.

13.5.3.4 `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.

13.5.3.5 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.

13.5.3.6 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.

13.5.3.7 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).

13.5.3.8 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

13.5.3.9 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

13.5.3.10 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

This function assigned a resource to a partition. This partition is then the owner. All resources always have an owner (one owner). The owner has various rights to make API calls affecting the resource. Ownership does not imply access to the peripheral itself (that is based on access rights).

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 defaulted 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

13.5.3.11 `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.

13.5.3.12 `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)

A subsystem is a physical grouping within the chip of related resources; this is SoC specific. This function is used to optimize moving resource for these groupings, for instance, an M4 core and its associated resources. The list of subsystems and associated resources can be found in the SoC-specific API document [Resources](#) chapter.

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.

13.5.3.13 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

Masters are IP blocks that generate bus transactions. This function configures how the isolation HW will define these bus transactions from the specified master. Note the security attribute will only be changed if the caller's partition is secure.

Note an IP block can be both a master and peripheral (have both a programming model and generate bus transactions).

13.5.3.14 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.

13.5.3.15 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*

Peripherals are IP blocks that have a programming model that can be accessed.

This function configures how the isolation HW will restrict access to a peripheral based on the attributes of a transaction from bus master. It also allows the access permissions of SC_R_SYSTEM to be set.

Note an IP block can be both a master and peripheral (have both a programming model and generate bus transactions).

13.5.3.16 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.

13.5.3.17 sc_rm_get_resource_owner()

```
sc_err_t sc_rm_get_resource_owner (
    sc_ipc_t ipc,
    sc_rsrc_t resource,
    sc_rm_pt_t * pt )
```

This function is used to get the owner of a resource.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>resource</i>	resource to check
out	<i>pt</i>	pointer to return owning partition

Returns

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

Return errors:

- SC_PARM if arguments out of range or invalid

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

13.5.3.18 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

Masters are IP blocks that generate bus transactions. Note an IP block can be both a master and peripheral (have both a programming model and generate bus transactions).

Returns

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

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

13.5.3.19 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

Peripherals are IP blocks that have a programming model that can be accessed. Note an IP block can be both a master and peripheral (have both a programming model and generate bus transactions)

Returns

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

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

13.5.3.20 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

13.5.3.21 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)

This function will create a new memory region. The area covered by the new region must already exist in a memory region owned by the caller. The result will be two memory regions, the new one overlapping the existing one. The new region has higher priority. See the XRDC2 MRC documentation for how it resolves access permissions in this case. By default, permissions will mirror the parent region.

13.5.3.22 `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 an existing 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_BUSY if the region is coincident with another region,
- SC_ERR_UNAVAILABLE if memory region table is full (no more allocation space)

This function will take an existing region and split it into two, non-overlapping regions. Note the new region must start or end on the split region. Permissions will mirror the parent region.

13.5.3.23 `sc_rm_memreg_frag()`

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

This function requests that the SC fragment a memory region.

Parameters

in	<i>ipc</i>	IPC handle
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_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_BUSY if the region is coincident with another region,
- SC_ERR_UNAVAILABLE if memory region table is full (no more allocation space)

This function finds the memory region containing the address range. It then splits it as required and returns the extracted region. The result is 2-3 non-overlapping regions, depending on how the new region aligns with existing regions. Permissions will mirror the parent region.

13.5.3.24 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

13.5.3.25 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.

13.5.3.26 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).

This function assigns a memory region to a partition. This partition is then the owner. All regions always have an owner (one owner). The owner has various rights to make API calls affecting the region. Ownership does not imply access to the memory itself (that is based on access rights).

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

13.5.3.27 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>

This operates on the memory region specified. If SC_RM_PT_ALL is specified then it operates on all the regions owned by the caller that exist at the time of the call.

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.

13.5.3.28 sc_rm_set_memreg_iee()

```
sc_err_t sc_rm_set_memreg_iee (
    sc_ipc_t ipc,
    sc_rm_mr_t mr,
    sc_rm_det_t det,
    sc_rm_rmsg_t rmsg )
```

This function configures the IEE parameters for a memory region.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mr</i>	handle of memory region to check
in	<i>det</i>	0 = normal, 1 = encrypted
in	<i>rmsg</i>	IEE region (0-7)

Caller must own SC_R_IEE_Rn where n is rmsg.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

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

13.5.3.29 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.

13.5.3.30 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

13.5.3.31 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

13.5.3.32 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)

This function assigned a pad to a partition. This partition is then the owner. All pads always have an owner (one owner). The owner has various rights to make API calls affecting the pad.

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.

13.5.3.33 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.

13.5.3.34 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
----	------------	------------

13.6 SECO: Security Service

Module for the Security (SECO) service.

Typedefs

- typedef [uint8_t sc_seco_auth_cmd_t](#)
This type is used to issue SECO authenticate commands.
- typedef [uint32_t sc_seco_rng_stat_t](#)
This type is used to return the RNG initialization status.

Defines for [sc_seco_auth_cmd_t](#)

- #define [SC_SECO_AUTH_CONTAINER](#) 0U
Authenticate container.
- #define [SC_SECO_VERIFY_IMAGE](#) 1U
Verify image.
- #define [SC_SECO_REL_CONTAINER](#) 2U
Release container.
- #define [SC_SECO_AUTH_SECO_FW](#) 3U
SECO Firmware.
- #define [SC_SECO_AUTH_HDMI_TX_FW](#) 4U
HDMI TX Firmware.
- #define [SC_SECO_AUTH_HDMI_RX_FW](#) 5U
HDMI RX Firmware.
- #define [SC_SECO_EVERIFY_IMAGE](#) 6U
Enhanced verify image.

Defines for [seco_rng_stat_t](#)

- #define [SC_SECO_RNG_STAT_UNAVAILABLE](#) 0U
Unable to initialize the RNG.
- #define [SC_SECO_RNG_STAT_INPROGRESS](#) 1U
Initialization is on-going.
- #define [SC_SECO_RNG_STAT_READY](#) 2U
Initialized.

Image Functions

- [sc_err_t sc_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_seco_authenticate](#) ([sc_ipc_t](#) ipc, [sc_seco_auth_cmd_t](#) cmd, [sc_faddr_t](#) addr)
This function is used to authenticate a SECO image or command.
- [sc_err_t sc_seco_enh_authenticate](#) ([sc_ipc_t](#) ipc, [sc_seco_auth_cmd_t](#) cmd, [sc_faddr_t](#) addr, [uint32_t](#) mask1, [uint32_t](#) mask2)
This function is used to authenticate a SECO image or command.

Lifecycle Functions

- `sc_err_t sc_seco_forward_lifecycle` (`sc_ipc_t ipc`, `uint32_t change`)
This function updates the lifecycle of the device.
- `sc_err_t sc_seco_return_lifecycle` (`sc_ipc_t ipc`, `sc_faddr_t addr`)
This function updates the lifecycle to one of the return lifecycles.
- `sc_err_t sc_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.

Attestation Functions

- `sc_err_t sc_seco_attest_mode` (`sc_ipc_t ipc`, `uint32_t mode`)
This function is used to set the attestation mode.
- `sc_err_t sc_seco_attest` (`sc_ipc_t ipc`, `uint64_t nonce`)
This function is used to request attestation.
- `sc_err_t sc_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_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_seco_attest_verify` (`sc_ipc_t ipc`, `sc_faddr_t addr`)
This function is used to verify attestation.

Key Functions

- `sc_err_t sc_seco_gen_key_blob` (`sc_ipc_t ipc`, `uint32_t id`, `sc_faddr_t load_addr`, `sc_faddr_t export_addr`, `uint16_t max_size`)
This function is used to generate a SECO key blob.
- `sc_err_t sc_seco_load_key` (`sc_ipc_t ipc`, `uint32_t id`, `sc_faddr_t addr`)
This function is used to load a SECO key.

Manufacturing Protection Functions

- `sc_err_t sc_seco_get_mp_key` (`sc_ipc_t ipc`, `sc_faddr_t dst_addr`, `uint16_t dst_size`)
This function is used to get the manufacturing protection public key.
- `sc_err_t sc_seco_update_mpmr` (`sc_ipc_t ipc`, `sc_faddr_t addr`, `uint8_t size`, `uint8_t lock`)
This function is used to update the manufacturing protection message register.
- `sc_err_t sc_seco_get_mp_sign` (`sc_ipc_t ipc`, `sc_faddr_t msg_addr`, `uint16_t msg_size`, `sc_faddr_t dst_addr`, `uint16_t dst_size`)
This function is used to get the manufacturing protection signature.

Debug Functions

- `void sc_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_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_seco_enable_debug (sc_ipc_t ipc, sc_faddr_t addr)`
This function securely enables debug.
- `sc_err_t sc_seco_get_event (sc_ipc_t ipc, uint8_t idx, uint32_t *event)`
This function is used to return an event from the SECO error log.

Miscellaneous Functions

- `sc_err_t sc_seco_fuse_write (sc_ipc_t ipc, sc_faddr_t addr)`
This function securely writes a group of fuse words.
- `sc_err_t sc_seco_patch (sc_ipc_t ipc, sc_faddr_t addr)`
This function applies a patch.
- `sc_err_t sc_seco_set_mono_counter_partition (sc_ipc_t ipc, uint16_t *she)`
This function partitions the monotonic counter.
- `sc_err_t sc_seco_set_fips_mode (sc_ipc_t ipc, uint8_t mode, uint32_t *reason)`
This function configures the SECO in FIPS mode.
- `sc_err_t sc_seco_start_rng (sc_ipc_t ipc, sc_seco_rng_stat_t *status)`
This function starts the random number generator.
- `sc_err_t sc_seco_sab_msg (sc_ipc_t ipc, sc_faddr_t addr)`
This function sends a generic signed message to the SECO SHE/HSM components.
- `sc_err_t sc_seco_secvio_enable (sc_ipc_t ipc)`
This function is used to enable security violation and tamper interrupts.
- `sc_err_t sc_seco_secvio_config (sc_ipc_t ipc, uint8_t id, uint8_t access, uint32_t *data0, uint32_t *data1, uint32_t *data2, uint32_t *data3, uint32_t *data4, uint8_t size)`
This function is used to read/write SNVS security violation and tamper registers.
- `sc_err_t sc_seco_secvio_dgo_config (sc_ipc_t ipc, uint8_t id, uint8_t access, uint32_t *data)`
This function is used to read/write SNVS security violation and tamper DGO registers.

13.6.1 Detailed Description

Module for the Security (SECO) service.

SECO functions in the SCFW API communicate via a messaging protocol to the Security Controller (SECO). This messaging protocol is documented in the *SECO API Reference Guide*. There is also a *Security Reference Manual (SRM)* that documents many of the SECO features.

SECO messages contain error codes defined in the *SECO API Reference Guide*. These have to be mapped to SC↔FW error codes. `sc_seco_get_event()` can be used to obtain SECO-specific error codes recorded in the SECO event log.

SCFW Error Mapping to SECO Errors

- SC_ERR_NOACCESS

- AHAB_INVALID_LIFECYCLE
- AHAB_PERMISSION_DENIED_IND
- SC_ERR_IPC
 - AHAB_INVALID_MESSAGE_IND
 - AHAB_CRC_ERROR
 - AHAB_INVALID_OPERATION_IND
- SC_ERR_VERSION
 - AHAB_BAD_VERSION_IND
- SC_ERR_PARM
 - AHAB_BAD_HASH_IND
 - AHAB_BAD_VALUE_IND
 - AHAB_BAD_FUSE_ID_IND
 - AHAB_BAD_CONTAINER_IND
 - AHAB_BAD_KEY_HASH_IND
 - AHAB_NO_VALID_CONTAINER_IND
 - AHAB_MUST_SIGNED_IND
 - AHAB_NO_BID_MATCHING_IND
 - AHAB_UNKNOWN_BID_IND
 - AHAB_UNALIGNED_PAYLOAD_IND
 - AHAB_WRONG_SIZE_IND
 - AHAB_OTP_INVALID_IDX_IND
 - AHAB_ADM_WRONG_LC_IND
 - AHAB_OTP_DED_IND
 - AHAB_BAD_PAYLOAD_IND
 - AHAB_WRONG_ADDRESS_IND
 - AHAB_DMA_FAILURE_IND
 - AHAB_BAD_UID_IND
 - AHAB_INCONSISTENT_PAR_IND
 - AHAB_BAD_MONOTONIC_COUNTER_IND
 - AHAB_BAD_SRK_SET_IND
 - AHAB_BAD_ID_IND
- SC_ERR_LOCKED
 - AHAB_OTP_LOCKED_IND
 - AHAB_LOCKED_REG_IND
- SC_ERR_UNAVAILABLE
 - AHAB_BID_COLLISION_IND
 - AHAB_OOM_FOR_BLOB_IND
 - AHAB_OOM_FOR_BLOB_EXPORT_IND
 - AHAB_HDCP_DISABLED_IND

- AHAB_NON_SECURE_STATE_IND
- AHAB_DISABLED_FEATURE_IND
- SC_ERR_BUSY
 - AHAB_ADM_NOT_READY_IND
 - AHAB_TIME_OUT_IND
- SC_ERR_FAIL
 - AHAB_BAD_BLOB_IND
 - AHAB_ENCRYPTION_FAILURE_IND
 - AHAB_DECRYPTION_FAILURE_IND
 - AHAB_BAD_CERTIFICATE_IND
 - AHAB_OTP_PROGFAIL_IND
 - AHAB_KMEK_GENERATION_FAIL_IND
 - AHAB_BAD_SIGNATURE_IND
 - AHAB_INVALID_KEY_IND
 - AHAB_MUST_ATTEST_IND
 - AHAB_RNG_NOT_STARTED_IND
 - AHAB_SECO_FATAL_FAILURE_IND
 - AHAB_RNG_INST_FAILURE_IND
 - AHAB_NO_AUTHENTICATION_IND
 - AHAB_AUTH_SKIPPED_OR_FAILED_IND
- If no mapping exists, SC_ERR_FAIL will be returned

13.6.2 Function Documentation

13.6.2.1 sc_seco_image_load()

```
sc_err_t sc_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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

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 *SECO API Reference Guide* for more info.

13.6.2.2 sc_seco_authenticate()

```
sc_err_t sc_seco_authenticate (
    sc_ipc_t ipc,
    sc_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,
- SC_ERR_BUSY if SECO is busy with another authentication request,
- SC_ERR_FAIL if SECO response is bad,
- SC_ERR_IPC if SECO response has bad header tag or size,

- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

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.

Implementation of this command depends on the underlying security architecture of the device. For example, on devices with SECO FW, the following options apply:

- cmd=SC_SECO_AUTH_CONTAINER, addr=container address (sends AHAB_AUTH_CONTAINER_REQ to SECO)
- cmd=SC_SECO_VERIFY_IMAGE, addr=image mask (sends AHAB_VERIFY_IMAGE_REQ to SECO)
- cmd=SC_SECO_REL_CONTAINER, addr unused (sends AHAB_RELEASE_CONTAINER_REQ to SECO)
- cmd=SC_SECO_AUTH_HDMI_TX_FW, addr unused (sends AHAB_ENABLE_HDMI_X_REQ with Subsystem=0 to SECO)
- cmd=SC_SECO_AUTH_HDMI_RX_FW, addr unused (sends AHAB_ENABLE_HDMI_X_REQ with Subsystem=1 to SECO)

See the *SECO API Reference Guide* for more info.

13.6.2.3 sc_seco_enh_authenticate()

```
sc_err_t sc_seco_enh_authenticate (
    sc_ipc_t ipc,
    sc_seco_auth_cmd_t cmd,
    sc_faddr_t addr,
    uint32_t mask1,
    uint32_t mask2 )
```

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

This is an enhanced version that has additional mask arguments.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>cmd</i>	authenticate command
in	<i>addr</i>	address of/or metadata
in	<i>mask1</i>	metadata
in	<i>mask2</i>	metadata

Return errors codes:

- SC_ERR_PARM if word fuse index param out of range or invalid,

- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_BUSY if SECO is busy with another authentication request,
- SC_ERR_FAIL if SECO response is bad,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

This supports all the commands found in [sc_seco_authenticate\(\)](#). Those commands should set both masks to 0 (except SC_SECO_VERIFY_IMAGE).

New commands are as follows:

- cmd=SC_SECO_VERIFY_IMAGE, addr unused, mask1=image mask, mask2 unused (sends AHAB_VERIFY↔_IMAGE_REQ to SECO)
- cmd=SC_SECO_EVERIFY_IMAGE, addr=container address, mask1=image mask, mask2=move mask (sends AHAB_EVERIFY_IMAGE_REQ to SECO)

See the *SECO API Reference Guide* for more info.

13.6.2.4 sc_seco_forward_lifecycle()

```
sc_err_t sc_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 transition

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

This function 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 *SECO API Reference Guide*.

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.

13.6.2.5 sc_seco_return_lifecycle()

```
sc_err_t sc_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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

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 *SECO API Reference Guide* for more info.

13.6.2.6 sc_seco_commit()

```
sc_err_t sc_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

<i>in</i>	<i>ipc</i>	IPC handle
<i>in, out</i>	<i>info</i>	pointer to information type to be committed

The return *info* 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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

13.6.2.7 sc_seco_attest_mode()

```
sc_err_t sc_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

<i>in</i>	<i>ipc</i>	IPC handle
<i>in</i>	<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_ATTESTATON not owned by caller,

- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

This is used to set the SECO attestation mode. This can be prover or verifier. See the *SECO API Reference Guide* for more on the supported modes, mode values, and mode behavior.

13.6.2.8 sc_seco_attest()

```
sc_err_t sc_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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

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 *SECO API Reference Guide* for more info.

13.6.2.9 sc_seco_get_attest_pkey()

```
sc_err_t sc_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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

See the *SECO API Reference Guide* for more info.

13.6.2.10 sc_seco_get_attest_sign()

```
sc_err_t sc_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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

See the *SECO API Reference Guide* for more info.

13.6.2.11 sc_seco_attest_verify()

```
sc_err_t sc_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,

- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

See the *SECO API Reference Guide* for more info.

13.6.2.12 sc_seco_gen_key_blob()

```
sc_err_t sc_seco_gen_key_blob (
    sc_ipc_t ipc,
    uint32_t id,
    sc_faddr_t load_addr,
    sc_faddr_t export_addr,
    uint16_t max_size )
```

This function is used to generate a SECO key blob.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>id</i>	key identifier
in	<i>load_addr</i>	load address
in	<i>export_addr</i>	export address
in	<i>max_size</i>	max export size

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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

This function is used to encapsulate sensitive keys in a specific structure called a blob, which provides both confidentiality and integrity protection.

See the *SECO API Reference Guide* for more info.

13.6.2.13 sc_seco_load_key()

```
sc_err_t sc_seco_load_key (
    sc_ipc_t ipc,
    uint32_t id,
    sc_faddr_t addr )
```

This function is used to load a SECO key.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>id</i>	key identifier
in	<i>addr</i>	key address

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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

This function is used to install private cryptographic keys encapsulated in a blob previously generated by SECO. The controller can be either the IEE or the VPU. The blob header carries the controller type and the key size, as provided by the user when generating the key blob.

See the *SECO API Reference Guide* for more info.

13.6.2.14 sc_seco_get_mp_key()

```
sc_err_t sc_seco_get_mp_key (
    sc_ipc_t ipc,
    sc_faddr_t dst_addr,
    uint16_t dst_size )
```

This function is used to get the manufacturing protection public key.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>dst_addr</i>	destination address
in	<i>dst_size</i>	destination size

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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

This function is supported only in OEM-closed lifecycle. It generates the mfg public key and stores it in a specific location in the secure memory.

See the *SECO API Reference Guide* for more info.

13.6.2.15 sc_seco_update_mpmr()

```
sc_err_t sc_seco_update_mpmr (
    sc_ipc_t ipc,
    sc_faddr_t addr,
    uint8_t size,
    uint8_t lock )
```

This function is used to update the manufacturing protection message register.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>addr</i>	data address
in	<i>size</i>	size
in	<i>lock</i>	lock_reg

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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,

- Others, see the [Security Service Detailed Description](#) section

This function is supported only in OEM-closed lifecycle. It updates the content of the MPMR (Manufacturing Protection Message register of 256 bits). This register will be appended to the input-data message when generating the signature. Please refer to the CAAM block guide for details.

See the *SECO API Reference Guide* for more info.

13.6.2.16 sc_seco_get_mp_sign()

```
sc_err_t sc_seco_get_mp_sign (
    sc_ipc_t ipc,
    sc_faddr_t msg_addr,
    uint16_t msg_size,
    sc_faddr_t dst_addr,
    uint16_t dst_size )
```

This function is used to get the manufacturing protection signature.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>msg_addr</i>	message address
in	<i>msg_size</i>	message size
in	<i>dst_addr</i>	destination address
in	<i>dst_size</i>	destination size

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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

This function is used to generate an ECDSA signature for an input-data message and to store it in a specific location in the secure memory. It is only supported in OEM-closed lifecycle. In order to get the ECDSA signature, the RNG must be initialized. In case it has not been started an error will be returned.

See the *SECO API Reference Guide* for more info.

13.6.2.17 `sc_seco_build_info()`

```
void sc_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)

13.6.2.18 `sc_seco_chip_info()`

```
sc_err_t sc_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)

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

13.6.2.19 sc_seco_enable_debug()

```
sc_err_t sc_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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

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

See the *SECO API Reference Guide* for more info.

13.6.2.20 sc_seco_get_event()

```
sc_err_t sc_seco_get_event (
    sc_ipc_t ipc,
    uint8_t idx,
    uint32_t * event )
```

This function is used to return an event from the SECO error log.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>idx</i>	index of event to return
out	<i>event</i>	pointer to return event

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

Read of *idx* 0 captures events from SECO. Loop starting with 0 until an error is returned to dump all events.

13.6.2.21 sc_seco_fuse_write()

```
sc_err_t sc_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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

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

See the *SECO API Reference Guide* for more info.

13.6.2.22 sc_seco_patch()

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

This function applies a patch.

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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

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

See the *SECO API Reference Guide* for more info.

13.6.2.23 sc_seco_set_mono_counter_partition()

```
sc_err_t sc_seco_set_mono_counter_partition (
    sc_ipc_t ipc,
    uint16_t * she )
```

This function partitions the monotonic counter.

Only the owner of the SC_R_SYSTEM resource or a partition with access permissions to SC_R_SYSTEM can do this.

Parameters

in	<i>ipc</i>	IPC handle
in, out	<i>she</i>	pointer to number of SHE bits

SECO uses an OTP monotonic counter to protect the SHE and HSM key-stores from roll-back attack. This function is used to define the number of monotonic counter bits allocated to SHE use. Two monotonic counter bits are used to store this information while the remaining bits are allocated to the HSM user. This function must be called before any SHE or HSM key stores are created in the system, otherwise the default configuration is applied. Returns the actual number of SHE bits.

If the partition has been already configured, any attempt to re-configure the SHE partition to a different value will result in a failure response.

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_NOACCESS if caller does not have SC_R_SYSTEM access
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

See the *SECO API Reference Guide* for more info.

13.6.2.24 sc_seco_set_fips_mode()

```
sc_err_t sc_seco_set_fips_mode (
    sc_ipc_t ipc,
    uint8_t mode,
    uint32_t * reason )
```

This function configures the SECO in FIPS mode.

Only the owner of the SC_R_SYSTEM resource or a partition with access permissions to SC_R_SYSTEM can do this.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>mode</i>	FIPS mode
out	<i>reason</i>	pointer to return failure reason

This function permanently configures the SECO in FIPS approved mode. When in FIPS approved mode the following services will be disabled and receive a failure response:

- Encrypted boot is not supported
- Attestation is not supported
- Manufacturing protection is not supported
- DTCP load
- SHE services are not supported
- Assign JR is not supported (all JRs owned by SECO)

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_NOACCESS if caller does not have SC_R_SYSTEM access
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

See the *SECO API Reference Guide* for more info.

13.6.2.25 sc_seco_start_rng()

```
sc_err_t sc_seco_start_rng (
    sc_ipc_t ipc,
    sc_seco_rng_stat_t * status )
```

This function starts the random number generator.

Parameters

in	<i>ipc</i>	IPC handle
out	<i>status</i>	pointer to return state of RNG

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

The RNG is started automatically after all CPUs are booted. This function can be used to start earlier and to check the status.

See the *SECO API Reference Guide* for more info.

13.6.2.26 sc_seco_sab_msg()

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

This function sends a generic signed message to the SECO SHE/HSM components.

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,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

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

See the *SECO API Reference Guide* for more info.

13.6.2.27 sc_seco_secvio_enable()

```
sc_err_t sc_seco_secvio_enable (  
    sc_ipc_t ipc )
```

This function is used to enable security violation and tamper interrupts.

These are then reported using the IRQ service via the SC_IRQ_SECVIO interrupt. Note it is automatically enabled at boot.

Parameters

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

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_NOACCESS if caller does not own SC_R_SECVIO,
- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_IPC if SECO response has bad header tag or size,

- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

The security violation interrupt is self-masking. Once it is cleared in the SNVS it must be re-enabled using this function.

13.6.2.28 sc_seco_secvio_config()

```
sc_err_t sc_seco_secvio_config (
    sc_ipc_t ipc,
    uint8_t id,
    uint8_t access,
    uint32_t * data0,
    uint32_t * data1,
    uint32_t * data2,
    uint32_t * data3,
    uint32_t * data4,
    uint8_t size )
```

This function is used to read/write SNVS security violation and tamper registers.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>id</i>	register ID
in	<i>access</i>	0=read, 1=write
in	<i>data0</i>	pointer to data to read or write
in	<i>data1</i>	pointer to data to read or write
in	<i>data2</i>	pointer to data to read or write
in	<i>data3</i>	pointer to data to read or write
in	<i>data4</i>	pointer to data to read or write
in	<i>size</i>	number of valid data words

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_NOACCESS if caller does not own SC_R_SECVIO,
- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

Unused data words can be passed a NULL pointer.

See AHAB_MANAGE_SNVS_REQ in the *SECO API Reference Guide* for more info.

13.6.2.29 sc_seco_secvio_dgo_config()

```
sc_err_t sc_seco_secvio_dgo_config (
    sc_ipc_t ipc,
    uint8_t id,
    uint8_t access,
    uint32_t * data )
```

This function is used to read/write SNVS security violation and tamper DGO registers.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>id</i>	register ID
in	<i>access</i>	0=read, 1=write
in	<i>data</i>	pointer to data to read or write

Returns

Returns and error code (SC_ERR_NONE = success).

Return errors codes:

- SC_ERR_NOACCESS if caller does not own SC_R_SECVIO,
- SC_ERR_UNAVAILABLE if SECO not available,
- SC_ERR_IPC if SECO response has bad header tag or size,
- SC_ERR_VERSION if SECO response has bad version,
- Others, see the [Security Service Detailed Description](#) section

See AHAB_MANAGE_SNVS_DGO_REQ in the *SECO API Reference Guide* for more info.

13.7 TIMER: 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.

Watchdog 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_set_wdog_window](#) ([sc_ipc_t](#) ipc, [sc_timer_wdog_time_t](#) window)
This function sets the watchdog window 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.

13.7.1 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.

13.7.2 Function Documentation

13.7.2.1 `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).

13.7.2.2 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).

13.7.2.3 sc_timer_set_wdog_window()

```
sc_err_t sc_timer_set_wdog_window (
    sc_ipc_t ipc,
    sc_timer_wdog_time_t window )
```

This function sets the watchdog window in milliseconds.

If not set then the window defaults to the 0. Once locked this value cannot be changed.

Parameters

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

Returns

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

Calling [sc_timer_ping_wdog\(\)](#) before the window time has expired will result in the watchdog action being taken.

13.7.2.4 `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).

Return errors:

- SC_ERR_NOACCESS if caller's partition is not isolated,
- SC_ERR_BUSY if already started

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

If the calling partition is not isolated then the wdog cannot be used. This is always the case if a non-secure partition is running on the same CPU as a secure partition (e.g. Linux under TZ). See [sc_rm_partition_alloc\(\)](#).

13.7.2.5 `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).

13.7.2.6 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).

13.7.2.7 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).

13.7.2.8 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).

13.7.2.9 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

13.7.2.10 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 or a partition with access permissions to SC_R_SYSTEM 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 cannot access SC_R_SYSTEM

13.7.2.11 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).

13.7.2.12 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 seconds

Returns

Returns an error code (SC_ERR_NONE = success).

13.7.2.13 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. This is an absolute time.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters

13.7.2.14 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).

Note this is a relative time.

Return errors:

- SC_ERR_PARM if invalid time/date parameters

13.7.2.15 sc_timer_cancel_rtc_alarm()

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

This function cancels the RTC 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

13.7.2.16 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 or a partition with access permissions to SC_R_SYSTEM can set the calibration.

Parameters

in	<i>ipc</i>	IPC handle
in	<i>count</i>	calibration 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 periodic basis, once per 32768 cycles of the RTC clock.

Returns

Returns an error code (SC_ERR_NONE = success).

13.7.2.17 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 the *ticks* parameter is an absolute time. 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

13.7.2.18 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

Note the *ticks* parameter is a relative time.

Returns

Returns an error code (SC_ERR_NONE = success).

Return errors:

- SC_ERR_PARM if invalid time/date parameters

13.7.2.19 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 14

File Documentation

14.1 platform/config/mx8qx/pads.h File Reference

Header file used to configure SoC pad list.

Macros

Pad Definitions

- `#define SC_P_PCIE_CTRL0_PERST_B 0`
HSIO.PCIE0.PERST_B, LSIO.GPIO4.IO00.
- `#define SC_P_PCIE_CTRL0_CLKREQ_B 1`
HSIO.PCIE0.CLKREQ_B, LSIO.GPIO4.IO01.
- `#define SC_P_PCIE_CTRL0_WAKE_B 2`
HSIO.PCIE0.WAKE_B, LSIO.GPIO4.IO02.
- `#define SC_P_COMP_CTL_GPIO_1V8_3V3_PCIESEP 3`
- `#define SC_P_USB_SS3_TC0 4`
ADMA.I2C1.SCL, CONN.USB_OTG1.PWR, CONN.USB_OTG2.PWR, LSIO.GPIO4.IO03.
- `#define SC_P_USB_SS3_TC1 5`
ADMA.I2C1.SCL, CONN.USB_OTG2.PWR, LSIO.GPIO4.IO04.
- `#define SC_P_USB_SS3_TC2 6`
ADMA.I2C1.SDA, CONN.USB_OTG1.OC, CONN.USB_OTG2.OC, LSIO.GPIO4.IO05.
- `#define SC_P_USB_SS3_TC3 7`
ADMA.I2C1.SDA, CONN.USB_OTG2.OC, LSIO.GPIO4.IO06.
- `#define SC_P_COMP_CTL_GPIO_3V3_USB3IO 8`
- `#define SC_P_EMMC0_CLK 9`
CONN.EMMC0.CLK, CONN.NAND.READY_B, LSIO.GPIO4.IO07.
- `#define SC_P_EMMC0_CMD 10`
CONN.EMMC0.CMD, CONN.NAND.DQS, LSIO.GPIO4.IO08.
- `#define SC_P_EMMC0_DATA0 11`
CONN.EMMC0.DATA0, CONN.NAND.DATA00, LSIO.GPIO4.IO09.
- `#define SC_P_EMMC0_DATA1 12`
CONN.EMMC0.DATA1, CONN.NAND.DATA01, LSIO.GPIO4.IO10.
- `#define SC_P_EMMC0_DATA2 13`
CONN.EMMC0.DATA2, CONN.NAND.DATA02, LSIO.GPIO4.IO11.
- `#define SC_P_EMMC0_DATA3 14`

- CONN.EMMC0.DATA3, CONN.NAND.DATA03, LSIO.GPIO4.IO12.
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_SD1FIX0** 15
- #define **SC_P_EMMC0_DATA4** 16
 - CONN.EMMC0.DATA4, CONN.NAND.DATA04, CONN.EMMC0.WP, LSIO.GPIO4.IO13.
- #define **SC_P_EMMC0_DATA5** 17
 - CONN.EMMC0.DATA5, CONN.NAND.DATA05, CONN.EMMC0.VSELECT, LSIO.GPIO4.IO14.
- #define **SC_P_EMMC0_DATA6** 18
 - CONN.EMMC0.DATA6, CONN.NAND.DATA06, CONN.MLB.CLK, LSIO.GPIO4.IO15.
- #define **SC_P_EMMC0_DATA7** 19
 - CONN.EMMC0.DATA7, CONN.NAND.DATA07, CONN.MLB.SIG, LSIO.GPIO4.IO16.
- #define **SC_P_EMMC0_STROBE** 20
 - CONN.EMMC0.STROBE, CONN.NAND.CLE, CONN.MLB.DATA, LSIO.GPIO4.IO17.
- #define **SC_P_EMMC0_RESET_B** 21
 - CONN.EMMC0.RESET_B, CONN.NAND.WP_B, LSIO.GPIO4.IO18.
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_SD1FIX1** 22
- #define **SC_P_USDHC1_RESET_B** 23
 - CONN.USDHC1.RESET_B, CONN.NAND.RE_N, ADMA.SPI2.SCK, LSIO.GPIO4.IO19.
- #define **SC_P_USDHC1_VSELECT** 24
 - CONN.USDHC1.VSELECT, CONN.NAND.RE_P, ADMA.SPI2.SDO, CONN.NAND.RE_B, LSIO.GPIO4.IO20.
- #define **SC_P_CTL_NAND_RE_P_N** 25
- #define **SC_P_USDHC1_WP** 26
 - CONN.USDHC1.WP, CONN.NAND.DQS_N, ADMA.SPI2.SDI, LSIO.GPIO4.IO21.
- #define **SC_P_USDHC1_CD_B** 27
 - CONN.USDHC1.CD_B, CONN.NAND.DQS_P, ADMA.SPI2.CS0, CONN.NAND.DQS, LSIO.GPIO4.IO22.
- #define **SC_P_CTL_NAND_DQS_P_N** 28
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_VSELSEP** 29
- #define **SC_P_USDHC1_CLK** 30
 - CONN.USDHC1.CLK, ADMA.UART3.RX, LSIO.GPIO4.IO23.
- #define **SC_P_USDHC1_CMD** 31
 - CONN.USDHC1.CMD, CONN.NAND.CE0_B, ADMA.MQS.R, LSIO.GPIO4.IO24.
- #define **SC_P_USDHC1_DATA0** 32
 - CONN.USDHC1.DATA0, CONN.NAND.CE1_B, ADMA.MQS.L, LSIO.GPIO4.IO25.
- #define **SC_P_USDHC1_DATA1** 33
 - CONN.USDHC1.DATA1, CONN.NAND.RE_B, ADMA.UART3.TX, LSIO.GPIO4.IO26.
- #define **SC_P_USDHC1_DATA2** 34
 - CONN.USDHC1.DATA2, CONN.NAND.WE_B, ADMA.UART3.CTS_B, LSIO.GPIO4.IO27.
- #define **SC_P_USDHC1_DATA3** 35
 - CONN.USDHC1.DATA3, CONN.NAND.ALE, ADMA.UART3.RTS_B, LSIO.GPIO4.IO28.
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_VSEL3** 36
- #define **SC_P_ENET0_RGMII_TXC** 37
 - CONN.ENET0.RGMII_TXC, CONN.ENET0.RCLK50M_OUT, CONN.ENET0.RCLK50M_IN, CONN.NAND.CE1_B, L←SIO.GPIO4.IO29.
- #define **SC_P_ENET0_RGMII_TX_CTL** 38
 - CONN.ENET0.RGMII_TX_CTL, CONN.USDHC1.RESET_B, LSIO.GPIO4.IO30.
- #define **SC_P_ENET0_RGMII_TXD0** 39
 - CONN.ENET0.RGMII_TXD0, CONN.USDHC1.VSELECT, LSIO.GPIO4.IO31.
- #define **SC_P_ENET0_RGMII_TXD1** 40
 - CONN.ENET0.RGMII_TXD1, CONN.USDHC1.WP, LSIO.GPIO5.IO00.
- #define **SC_P_ENET0_RGMII_TXD2** 41
 - CONN.ENET0.RGMII_TXD2, CONN.MLB.CLK, CONN.NAND.CE0_B, CONN.USDHC1.CD_B, LSIO.GPIO5.IO01.
- #define **SC_P_ENET0_RGMII_TXD3** 42
 - CONN.ENET0.RGMII_TXD3, CONN.MLB.SIG, CONN.NAND.RE_B, LSIO.GPIO5.IO02.
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_ENET_ENETB0** 43
- #define **SC_P_ENET0_RGMII_RXC** 44
 - CONN.ENET0.RGMII_RXC, CONN.MLB.DATA, CONN.NAND.WE_B, CONN.USDHC1.CLK, LSIO.GPIO5.IO03.
- #define **SC_P_ENET0_RGMII_RX_CTL** 45

- CONN.ENET0.RGMII_RX_CTL, CONN.USDHC1.CMD, LSIO.GPIO5.IO04.
- #define [SC_P_ENET0_RGMII_RXD0](#) 46
CONN.ENET0.RGMII_RXD0, CONN.USDHC1.DATA0, LSIO.GPIO5.IO05.
- #define [SC_P_ENET0_RGMII_RXD1](#) 47
CONN.ENET0.RGMII_RXD1, CONN.USDHC1.DATA1, LSIO.GPIO5.IO06.
- #define [SC_P_ENET0_RGMII_RXD2](#) 48
CONN.ENET0.RGMII_RXD2, CONN.ENET0.RMII_RX_ER, CONN.USDHC1.DATA2, LSIO.GPIO5.IO07.
- #define [SC_P_ENET0_RGMII_RXD3](#) 49
CONN.ENET0.RGMII_RXD3, CONN.NAND.ALE, CONN.USDHC1.DATA3, LSIO.GPIO5.IO08.
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_ENET_ENETB1** 50
- #define [SC_P_ENET0_REFCLK_125M_25M](#) 51
CONN.ENET0.REFCLK_125M_25M, CONN.ENET0.PPS, CONN.ENET1.PPS, LSIO.GPIO5.IO09.
- #define [SC_P_ENET0_MDIO](#) 52
CONN.ENET0.MDIO, ADMA.I2C3.SDA, CONN.ENET1.MDIO, LSIO.GPIO5.IO10.
- #define [SC_P_ENET0_MDC](#) 53
CONN.ENET0.MDC, ADMA.I2C3.SCL, CONN.ENET1.MDC, LSIO.GPIO5.IO11.
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_GPIOCT** 54
- #define [SC_P_ESAIO_FSR](#) 55
ADMA.ESAIO.FSR, CONN.ENET1.RCLK50M_OUT, ADMA.LCDIF.D00, CONN.ENET1.RGMII_TXC, CONN.ENE←
T1.RCLK50M_IN.
- #define [SC_P_ESAIO_FST](#) 56
ADMA.ESAIO.FST, CONN.MLB.CLK, ADMA.LCDIF.D01, CONN.ENET1.RGMII_TXD2, LSIO.GPIO0.IO01.
- #define [SC_P_ESAIO_SCKR](#) 57
ADMA.ESAIO.SCKR, ADMA.LCDIF.D02, CONN.ENET1.RGMII_TX_CTL, LSIO.GPIO0.IO02.
- #define [SC_P_ESAIO_SCKT](#) 58
ADMA.ESAIO.SCKT, CONN.MLB.SIG, ADMA.LCDIF.D03, CONN.ENET1.RGMII_TXD3, LSIO.GPIO0.IO03.
- #define [SC_P_ESAIO_TX0](#) 59
ADMA.ESAIO.TX0, CONN.MLB.DATA, ADMA.LCDIF.D04, CONN.ENET1.RGMII_RXC, LSIO.GPIO0.IO04.
- #define [SC_P_ESAIO_TX1](#) 60
ADMA.ESAIO.TX1, ADMA.LCDIF.D05, CONN.ENET1.RGMII_RXD3, LSIO.GPIO0.IO05.
- #define [SC_P_ESAIO_TX2_RX3](#) 61
ADMA.ESAIO.TX2_RX3, CONN.ENET1.RMII_RX_ER, ADMA.LCDIF.D06, CONN.ENET1.RGMII_RXD2, LSIO.GPI←
O0.IO06.
- #define [SC_P_ESAIO_TX3_RX2](#) 62
ADMA.ESAIO.TX3_RX2, ADMA.LCDIF.D07, CONN.ENET1.RGMII_RXD1, LSIO.GPIO0.IO07.
- #define [SC_P_ESAIO_TX4_RX1](#) 63
ADMA.ESAIO.TX4_RX1, ADMA.LCDIF.D08, CONN.ENET1.RGMII_TXD0, LSIO.GPIO0.IO08.
- #define [SC_P_ESAIO_TX5_RX0](#) 64
ADMA.ESAIO.TX5_RX0, ADMA.LCDIF.D09, CONN.ENET1.RGMII_TXD1, LSIO.GPIO0.IO09.
- #define [SC_P_SPDIF0_RX](#) 65
ADMA.SPDIF0.RX, ADMA.MQS.R, ADMA.LCDIF.D10, CONN.ENET1.RGMII_RXD0, LSIO.GPIO0.IO10.
- #define [SC_P_SPDIF0_TX](#) 66
ADMA.SPDIF0.TX, ADMA.MQS.L, ADMA.LCDIF.D11, CONN.ENET1.RGMII_RX_CTL, LSIO.GPIO0.IO11.
- #define [SC_P_SPDIF0_EXT_CLK](#) 67
ADMA.SPDIF0.EXT_CLK, ADMA.LCDIF.D12, CONN.ENET1.REFCLK_125M_25M, LSIO.GPIO0.IO12.
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHB** 68
- #define [SC_P_SPI3_SCK](#) 69
ADMA.SPI3.SCK, ADMA.LCDIF.D13, LSIO.GPIO0.IO13.
- #define [SC_P_SPI3_SDO](#) 70
ADMA.SPI3.SDO, ADMA.LCDIF.D14, LSIO.GPIO0.IO14.
- #define [SC_P_SPI3_SDI](#) 71
ADMA.SPI3.SDI, ADMA.LCDIF.D15, LSIO.GPIO0.IO15.
- #define [SC_P_SPI3_CS0](#) 72
ADMA.SPI3.CS0, ADMA.ACM.MCLK_OUT1, ADMA.LCDIF.HSYNC, LSIO.GPIO0.IO16.
- #define [SC_P_SPI3_CS1](#) 73

- ADMA.SPI3.CS1, ADMA.I2C3.SCL, ADMA.LCDIF.RESET, ADMA.SPI2.CS0, ADMA.LCDIF.D16.
- #define [SC_P_MCLK_IN1](#) 74
ADMA.ACM.MCLK_IN1, ADMA.I2C3.SDA, ADMA.LCDIF.EN, ADMA.SPI2.SCK, ADMA.LCDIF.D17.
- #define [SC_P_MCLK_IN0](#) 75
ADMA.ACM.MCLK_IN0, ADMA.ESAI0.RX_HF_CLK, ADMA.LCDIF.VSYNC, ADMA.SPI2.SDI, LSIO.GPIO0.IO19.
- #define [SC_P_MCLK_OUT0](#) 76
ADMA.ACM.MCLK_OUT0, ADMA.ESAI0.TX_HF_CLK, ADMA.LCDIF.CLK, ADMA.SPI2.SDO, LSIO.GPIO0.IO20.
- #define [SC_P_UART1_TX](#) 77
ADMA.UART1.TX, LSIO.PWM0.OUT, LSIO.GPT0.CAPTURE, LSIO.GPIO0.IO21.
- #define [SC_P_UART1_RX](#) 78
ADMA.UART1.RX, LSIO.PWM1.OUT, LSIO.GPT0.COMPARE, LSIO.GPT1.CLK, LSIO.GPIO0.IO22.
- #define [SC_P_UART1_RTS_B](#) 79
ADMA.UART1.RTS_B, LSIO.PWM2.OUT, ADMA.LCDIF.D16, LSIO.GPT1.CAPTURE, LSIO.GPT0.CLK.
- #define [SC_P_UART1_CTS_B](#) 80
ADMA.UART1.CTS_B, LSIO.PWM3.OUT, ADMA.LCDIF.D17, LSIO.GPT1.COMPARE, LSIO.GPIO0.IO24.
- #define [SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHK](#) 81
- #define [SC_P_SAI0_TXD](#) 82
ADMA.SAI0.TXD, ADMA.SAI1.RXC, ADMA.SPI1.SDO, ADMA.LCDIF.D18, LSIO.GPIO0.IO25.
- #define [SC_P_SAI0_TXC](#) 83
ADMA.SAI0.TXC, ADMA.SAI1.TXD, ADMA.SPI1.SDI, ADMA.LCDIF.D19, LSIO.GPIO0.IO26.
- #define [SC_P_SAI0_RXD](#) 84
ADMA.SAI0.RXD, ADMA.SAI1.RXFS, ADMA.SPI1.CS0, ADMA.LCDIF.D20, LSIO.GPIO0.IO27.
- #define [SC_P_SAI0_TXFS](#) 85
ADMA.SAI0.TXFS, ADMA.SPI2.CS1, ADMA.SPI1.SCK, LSIO.GPIO0.IO28.
- #define [SC_P_SAI1_RXD](#) 86
ADMA.SAI1.RXD, ADMA.SAI0.RXFS, ADMA.SPI1.CS1, ADMA.LCDIF.D21, LSIO.GPIO0.IO29.
- #define [SC_P_SAI1_RXC](#) 87
ADMA.SAI1.RXC, ADMA.SAI1.TXC, ADMA.LCDIF.D22, LSIO.GPIO0.IO30.
- #define [SC_P_SAI1_RXFS](#) 88
ADMA.SAI1.RXFS, ADMA.SAI1.TXFS, ADMA.LCDIF.D23, LSIO.GPIO0.IO31.
- #define [SC_P_SPI2_CS0](#) 89
ADMA.SPI2.CS0, LSIO.GPIO1.IO00.
- #define [SC_P_SPI2_SDO](#) 90
ADMA.SPI2.SDO, LSIO.GPIO1.IO01.
- #define [SC_P_SPI2_SDI](#) 91
ADMA.SPI2.SDI, LSIO.GPIO1.IO02.
- #define [SC_P_SPI2_SCK](#) 92
ADMA.SPI2.SCK, LSIO.GPIO1.IO03.
- #define [SC_P_SPI0_SCK](#) 93
ADMA.SPI0.SCK, ADMA.SAI0.TXC, M40.I2C0.SCL, M40.GPIO0.IO00, LSIO.GPIO1.IO04.
- #define [SC_P_SPI0_SDI](#) 94
ADMA.SPI0.SDI, ADMA.SAI0.TXD, M40.TPM0.CH0, M40.GPIO0.IO02, LSIO.GPIO1.IO05.
- #define [SC_P_SPI0_SDO](#) 95
ADMA.SPI0.SDO, ADMA.SAI0.TXFS, M40.I2C0.SDA, M40.GPIO0.IO01, LSIO.GPIO1.IO06.
- #define [SC_P_SPI0_CS1](#) 96
ADMA.SPI0.CS1, ADMA.SAI0.RXC, ADMA.SAI1.TXD, ADMA.LCD_PWM0.OUT, LSIO.GPIO1.IO07.
- #define [SC_P_SPI0_CS0](#) 97
ADMA.SPI0.CS0, ADMA.SAI0.RXD, M40.TPM0.CH1, M40.GPIO0.IO03, LSIO.GPIO1.IO08.
- #define [SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHT](#) 98
- #define [SC_P_ADC_IN1](#) 99
ADMA.ADC.IN1, M40.I2C0.SDA, M40.GPIO0.IO01, LSIO.GPIO1.IO09.
- #define [SC_P_ADC_IN0](#) 100
ADMA.ADC.IN0, M40.I2C0.SCL, M40.GPIO0.IO00, LSIO.GPIO1.IO10.
- #define [SC_P_ADC_IN3](#) 101
ADMA.ADC.IN3, M40.UART0.TX, M40.GPIO0.IO03, ADMA.ACM.MCLK_OUT0, LSIO.GPIO1.IO11.

- #define [SC_P_ADC_IN2](#) 102
ADMA.ADC.IN2, M40.UART0.RX, M40.GPIO0.IO02, ADMA.ACM.MCLK_IN0, LSIO.GPIO1.IO12.
- #define [SC_P_ADC_IN5](#) 103
ADMA.ADC.IN5, M40.TPM0.CH1, M40.GPIO0.IO05, LSIO.GPIO1.IO13.
- #define [SC_P_ADC_IN4](#) 104
ADMA.ADC.IN4, M40.TPM0.CH0, M40.GPIO0.IO04, LSIO.GPIO1.IO14.
- #define [SC_P_FLEXCAN0_RX](#) 105
ADMA.FLEXCAN0.RX, ADMA.SAI2.RXC, ADMA.UART0.RTS_B, ADMA.SAI1.TXC, LSIO.GPIO1.IO15.
- #define [SC_P_FLEXCAN0_TX](#) 106
ADMA.FLEXCAN0.TX, ADMA.SAI2.RXD, ADMA.UART0.CTS_B, ADMA.SAI1.TXFS, LSIO.GPIO1.IO16.
- #define [SC_P_FLEXCAN1_RX](#) 107
ADMA.FLEXCAN1.RX, ADMA.SAI2.RXFS, ADMA.FTM.CH2, ADMA.SAI1.TXD, LSIO.GPIO1.IO17.
- #define [SC_P_FLEXCAN1_TX](#) 108
ADMA.FLEXCAN1.TX, ADMA.SAI3.RXC, ADMA.DMA0.REQ_IN0, ADMA.SAI1.RXD, LSIO.GPIO1.IO18.
- #define [SC_P_FLEXCAN2_RX](#) 109
ADMA.FLEXCAN2.RX, ADMA.SAI3.RXD, ADMA.UART3.RX, ADMA.SAI1.RXFS, LSIO.GPIO1.IO19.
- #define [SC_P_FLEXCAN2_TX](#) 110
ADMA.FLEXCAN2.TX, ADMA.SAI3.RXFS, ADMA.UART3.TX, ADMA.SAI1.RXC, LSIO.GPIO1.IO20.
- #define [SC_P_UART0_RX](#) 111
ADMA.UART0.RX, ADMA.MQS.R, ADMA.FLEXCAN0.RX, SCU.UART0.RX, LSIO.GPIO1.IO21.
- #define [SC_P_UART0_TX](#) 112
ADMA.UART0.TX, ADMA.MQS.L, ADMA.FLEXCAN0.TX, SCU.UART0.TX, LSIO.GPIO1.IO22.
- #define [SC_P_UART2_TX](#) 113
ADMA.UART2.TX, ADMA.FTM.CH1, ADMA.FLEXCAN1.TX, LSIO.GPIO1.IO23.
- #define [SC_P_UART2_RX](#) 114
ADMA.UART2.RX, ADMA.FTM.CH0, ADMA.FLEXCAN1.RX, LSIO.GPIO1.IO24.
- #define [SC_P_COMP_CTL_GPIO_1V8_3V3_GPIOLH](#) 115
- #define [SC_P_MIPIDSI0_I2C0_SCL](#) 116
MIPIDSI0.I2C0_SCL, MIPIDSI1.GPIO0.IO02, LSIO.GPIO1.IO25.
- #define [SC_P_MIPIDSI0_I2C0_SDA](#) 117
MIPIDSI0.I2C0_SDA, MIPIDSI1.GPIO0.IO03, LSIO.GPIO1.IO26.
- #define [SC_P_MIPIDSI0_GPIO0_00](#) 118
MIPIDSI0.GPIO0.IO00, ADMA.I2C1_SCL, MIPIDSI0.PWM0.OUT, LSIO.GPIO1.IO27.
- #define [SC_P_MIPIDSI0_GPIO0_01](#) 119
MIPIDSI0.GPIO0.IO01, ADMA.I2C1_SDA, LSIO.GPIO1.IO28.
- #define [SC_P_MIPIDSI1_I2C0_SCL](#) 120
MIPIDSI1.I2C0_SCL, MIPIDSI0.GPIO0.IO02, LSIO.GPIO1.IO29.
- #define [SC_P_MIPIDSI1_I2C0_SDA](#) 121
MIPIDSI1.I2C0_SDA, MIPIDSI0.GPIO0.IO03, LSIO.GPIO1.IO30.
- #define [SC_P_MIPIDSI1_GPIO0_00](#) 122
MIPIDSI1.GPIO0.IO00, ADMA.I2C2_SCL, MIPIDSI1.PWM0.OUT, LSIO.GPIO1.IO31.
- #define [SC_P_MIPIDSI1_GPIO0_01](#) 123
MIPIDSI1.GPIO0.IO01, ADMA.I2C2_SDA, LSIO.GPIO2.IO00.
- #define [SC_P_COMP_CTL_GPIO_1V8_3V3_MIPIDSIGPIO](#) 124
- #define [SC_P_JTAG_TRST_B](#) 125
SCU.JTAG.TRST_B, SCU.WDOG0.WDOG_OUT.
- #define [SC_P_PMIC_I2C_SCL](#) 126
SCU.PMIC_I2C_SCL, SCU.GPIO0.IOXX_PMIC_A35_ON, LSIO.GPIO2.IO01.
- #define [SC_P_PMIC_I2C_SDA](#) 127
SCU.PMIC_I2C_SDA, SCU.GPIO0.IOXX_PMIC_GPU_ON, LSIO.GPIO2.IO02.
- #define [SC_P_PMIC_INT_B](#) 128
SCU.DSC.PMIC_INT_B.
- #define [SC_P_SCU_GPIO0_00](#) 129
SCU.GPIO0.IO00, SCU.UART0.RX, M40.UART0.RX, ADMA.UART3.RX, LSIO.GPIO2.IO03.
- #define [SC_P_SCU_GPIO0_01](#) 130

- SCU.GPIO0.IO01, SCU.UART0.TX, M40.UART0.TX, ADMA.UART3.TX, SCU.WDOG0.WDOG_OUT.*
- #define **SC_P_SCU_PMIC_STANDBY** 131
 - SCU.DSC.PMIC_STANDBY.*
- #define **SC_P_SCU_BOOT_MODE0** 132
 - SCU.DSC.BOOT_MODE0.*
- #define **SC_P_SCU_BOOT_MODE1** 133
 - SCU.DSC.BOOT_MODE1.*
- #define **SC_P_SCU_BOOT_MODE2** 134
 - SCU.DSC.BOOT_MODE2, SCU.PMIC_I2C.SDA.*
- #define **SC_P_SCU_BOOT_MODE3** 135
 - SCU.DSC.BOOT_MODE3, SCU.PMIC_I2C.SCL, SCU.DSC.RTC_CLOCK_OUTPUT_32K.*
- #define **SC_P_CSI_D00** 136
 - CI_PI.D02, ADMA.SAI0.RXC.*
- #define **SC_P_CSI_D01** 137
 - CI_PI.D03, ADMA.SAI0.RXD.*
- #define **SC_P_CSI_D02** 138
 - CI_PI.D04, ADMA.SAI0.RXFS.*
- #define **SC_P_CSI_D03** 139
 - CI_PI.D05, ADMA.SAI2.RXC.*
- #define **SC_P_CSI_D04** 140
 - CI_PI.D06, ADMA.SAI2.RXD.*
- #define **SC_P_CSI_D05** 141
 - CI_PI.D07, ADMA.SAI2.RXFS.*
- #define **SC_P_CSI_D06** 142
 - CI_PI.D08, ADMA.SAI3.RXC.*
- #define **SC_P_CSI_D07** 143
 - CI_PI.D09, ADMA.SAI3.RXD.*
- #define **SC_P_CSI_HSYNC** 144
 - CI_PI.HSYNC, CI_PI.D00, ADMA.SAI3.RXFS.*
- #define **SC_P_CSI_VSYNC** 145
 - CI_PI.VSYNC, CI_PI.D01.*
- #define **SC_P_CSI_PCLK** 146
 - CI_PI.PCLK, MIPI_CSI0.I2C0.SCL, ADMA.SPI1.SCK, LSIO.GPIO3.IO00.*
- #define **SC_P_CSI_MCLK** 147
 - CI_PI.MCLK, MIPI_CSI0.I2C0.SDA, ADMA.SPI1.SDO, LSIO.GPIO3.IO01.*
- #define **SC_P_CSI_EN** 148
 - CI_PI.EN, CI_PI.I2C.SCL, ADMA.I2C3.SCL, ADMA.SPI1.SDI, LSIO.GPIO3.IO02.*
- #define **SC_P_CSI_RESET** 149
 - CI_PI.RESET, CI_PI.I2C.SDA, ADMA.I2C3.SDA, ADMA.SPI1.CS0, LSIO.GPIO3.IO03.*
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_GPIORHD** 150
- #define **SC_P_MIPI_CSI0_MCLK_OUT** 151
 - MIPI_CSI0.ACM.MCLK_OUT, LSIO.GPIO3.IO04.*
- #define **SC_P_MIPI_CSI0_I2C0_SCL** 152
 - MIPI_CSI0.I2C0.SCL, MIPI_CSI0.GPIO0.IO02, LSIO.GPIO3.IO05.*
- #define **SC_P_MIPI_CSI0_I2C0_SDA** 153
 - MIPI_CSI0.I2C0.SDA, MIPI_CSI0.GPIO0.IO03, LSIO.GPIO3.IO06.*
- #define **SC_P_MIPI_CSI0_GPIO0_01** 154
 - MIPI_CSI0.GPIO0.IO01, ADMA.I2C0.SDA, LSIO.GPIO3.IO07.*
- #define **SC_P_MIPI_CSI0_GPIO0_00** 155
 - MIPI_CSI0.GPIO0.IO00, ADMA.I2C0.SCL, LSIO.GPIO3.IO08.*
- #define **SC_P_QSPI0A_DATA0** 156
 - LSIO.QSPI0A.DATA0, LSIO.GPIO3.IO09.*
- #define **SC_P_QSPI0A_DATA1** 157
 - LSIO.QSPI0A.DATA1, LSIO.GPIO3.IO10.*
- #define **SC_P_QSPI0A_DATA2** 158

- LSIO.QSPI0A.DATA2, LSIO.GPIO3.IO11.*
- #define [SC_P_QSPI0A_DATA3](#) 159
 - LSIO.QSPI0A.DATA3, LSIO.GPIO3.IO12.*
- #define [SC_P_QSPI0A_DQS](#) 160
 - LSIO.QSPI0A.DQS, LSIO.GPIO3.IO13.*
- #define [SC_P_QSPI0A_SS0_B](#) 161
 - LSIO.QSPI0A.SS0_B, LSIO.GPIO3.IO14.*
- #define [SC_P_QSPI0A_SS1_B](#) 162
 - LSIO.QSPI0A.SS1_B, LSIO.GPIO3.IO15.*
- #define [SC_P_QSPI0A_SCLK](#) 163
 - LSIO.QSPI0A.SCLK, LSIO.GPIO3.IO16.*
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_QSPI0A** 164
- #define [SC_P_QSPI0B_SCLK](#) 165
 - LSIO.QSPI0B.SCLK, LSIO.QSPI1A.SCLK, LSIO.KPP0.COL0, LSIO.GPIO3.IO17.*
- #define [SC_P_QSPI0B_DATA0](#) 166
 - LSIO.QSPI0B.DATA0, LSIO.QSPI1A.DATA0, LSIO.KPP0.COL1, LSIO.GPIO3.IO18.*
- #define [SC_P_QSPI0B_DATA1](#) 167
 - LSIO.QSPI0B.DATA1, LSIO.QSPI1A.DATA1, LSIO.KPP0.COL2, LSIO.GPIO3.IO19.*
- #define [SC_P_QSPI0B_DATA2](#) 168
 - LSIO.QSPI0B.DATA2, LSIO.QSPI1A.DATA2, LSIO.KPP0.COL3, LSIO.GPIO3.IO20.*
- #define [SC_P_QSPI0B_DATA3](#) 169
 - LSIO.QSPI0B.DATA3, LSIO.QSPI1A.DATA3, LSIO.KPP0.ROW0, LSIO.GPIO3.IO21.*
- #define [SC_P_QSPI0B_DQS](#) 170
 - LSIO.QSPI0B.DQS, LSIO.QSPI1A.DQS, LSIO.KPP0.ROW1, LSIO.GPIO3.IO22.*
- #define [SC_P_QSPI0B_SS0_B](#) 171
 - LSIO.QSPI0B.SS0_B, LSIO.QSPI1A.SS0_B, LSIO.KPP0.ROW2, LSIO.GPIO3.IO23.*
- #define [SC_P_QSPI0B_SS1_B](#) 172
 - LSIO.QSPI0B.SS1_B, LSIO.QSPI1A.SS1_B, LSIO.KPP0.ROW3, LSIO.GPIO3.IO24.*
- #define **SC_P_COMP_CTL_GPIO_1V8_3V3_QSPI0B** 173

14.1.1 Detailed Description

Header file used to configure SoC pad list.

14.2 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.*

14.2.1 Detailed Description

Header file for the IPC implementation.

14.2.2 Function Documentation

14.2.2.1 `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.

14.2.2.2 `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
----	------------	------------------------

14.2.2.3 `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.

14.2.2.4 `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.

14.3 platform/main/types.h File Reference

Header file containing types used across multiple service APIs.

Macros

- #define `SC_R_NONE` 0xFFFF0U
Define for ATF/Linux.
- #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_OFS_AUDIO_ALT** 56U
- #define **SC_C_DSP_BYP** 57U
- #define **SC_C_CLK_GEN_EN** 58U
- #define **SC_C_INTF_SEL** 59U
- #define **SC_C_RXC_DLY** 60U
- #define **SC_C_TIMER_SEL** 61U
- #define **SC_C_MISC0** 62U
- #define **SC_C_MISC1** 63U
- #define **SC_C_MISC2** 64U
- #define **SC_C_MISC3** 65U
- #define **SC_C_LAST** 66U

- #define `SC_P_ALL` ((`sc_pad_t`) UINT16_MAX)
Define for used to specify all pads.

Defines for chip IDs

- #define `CHIP_ID_QM` 0x1U
i.MX8QM
- #define `CHIP_ID_QX` 0x2U
i.MX8QX
- #define `CHIP_ID_DXL` 0xEU
i.MX8DXL

Defines for common frequencies

- #define `SC_32KHZ` 32768U
32KHz
- #define `SC_1MHZ` 1000000U
1MHz
- #define `SC_10MHZ` 10000000U
10MHz
- #define `SC_16MHZ` 16000000U
16MHz
- #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_114MHZ` 114000000U
114MHz
- #define `SC_125MHZ` 125000000U
125MHz

- #define SC_128MHZ 128000000U
128MHz
- #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_465MHZ 465000000U
465MHz
- #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_1860MHZ](#) 1860000000U
1.86GHz
- #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_228MHZ](#) 228000000U
233MHz
- #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_456MHZ](#) 456000000U
466MHz
- #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_912MHZ](#) 912000000U
- 912MHz
- #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_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](#) 7U
- 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_PERF** 23U
- #define **SC_R_USB_1_PHY** 24U
- #define **SC_R_DC_0_WARP** 25U
- #define **SC_R_V2X_MU_0** 26U
- #define **SC_R_V2X_MU_1** 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_V2X_MU_2** 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_V2X_MU_3 40U
- #define SC_R_V2X_MU_4 41U
- #define SC_R_DC_1_WARP 42U
- #define SC_R_UNUSED1 43U
- #define SC_R_SECVIO 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_UNUSED13 48U
- #define SC_R_DC_1 49U
- #define SC_R_UNUSED14 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_V2X_PID0 140U
- #define SC_R_V2X_PID1 141U
- #define SC_R_V2X_PID2 142U
- #define SC_R_V2X_PID3 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_ENET_0_A0 290U
- #define SC_R_ENET_0_A1 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_UNUSED17 310U
- #define SC_R_UNUSED18 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_ENET_0_A2 366U
- #define SC_R_ENET_1_A0 367U
- #define SC_R_ENET_1_A1 368U
- #define SC_R_ENET_1_A2 369U
- #define SC_R_ENET_1_A3 370U
- #define SC_R_ENET_1_A4 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 **uint32_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.

14.3.1 Detailed Description

Header file containing types used across multiple service APIs.

14.3.2 Macro Definition Documentation

14.3.2.1 SC_R_NONE

```
#define SC_R_NONE 0xFFFF0U
```

Define for ATF/Linux.

Not used by SCFW. Not a valid parameter for any SCFW API calls!

14.3.2.2 SC_P_ALL

```
#define SC_P_ALL ((sc\_pad\_t) UINT16_MAX)
```

Define for used to specify all pads.

All pads

14.3.3 Typedef Documentation

14.3.3.1 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).

14.3.3.2 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.

14.4 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](#) 7U
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.
- #define [SC_IRQ_GROUP_REBOOTED](#) 5U
Partition reboot complete.
- #define [SC_IRQ_GROUP_REBOOT](#) 6U
Partition reboot starting.

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.
- #define `SC_IRQ_USR1` (1U << 2U)
User defined 1.

- #define `SC_IRQ_USR2` (1U << 3U)
User defined 2.
- #define `SC_IRQ_BC_PAD` (1U << 4U)
Pad wakeup (broadcast to all partitions)
- #define `SC_IRQ_SW_WAKE` (1U << 5U)
Software requested wake.
- #define `SC_IRQ_SECVIO` (1U << 6U)
Security violation.

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).

14.4.1 Detailed Description

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

14.5 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_bt_t`

- #define [SC_MISC_BT_PRIMARY](#) 0U
Primary boot.
- #define [SC_MISC_BT_SECONDARY](#) 1U
Secondary boot.
- #define [SC_MISC_BT_RECOVERY](#) 2U
Recovery boot.
- #define [SC_MISC_BT_MANUFACTURE](#) 3U
Manufacture boot.
- #define [SC_MISC_BT_SERIAL](#) 4U
Serial boot.

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_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.

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_api_ver](#) (sc_ipc_t ipc, [uint16_t](#) *cl_maj, [uint16_t](#) *cl_min, [uint16_t](#) *sv_maj, [uint16_t](#) *sv_min)
This function is used to return the SCFW API versions.
- 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.
- [sc_err_t sc_misc_get_boot_container](#) (sc_ipc_t ipc, [uint8_t](#) *idx)
This function returns the boot container index.
- 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.
- [sc_err_t sc_misc_board_ioctl](#) (sc_ipc_t ipc, [uint32_t](#) *parm1, [uint32_t](#) *parm2, [uint32_t](#) *parm3)
This function calls the board IOCTL function.

14.5.1 Detailed Description

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

14.6 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.
- [sc_err_t sc_pad_config](#) (sc_ipc_t ipc, [sc_pad_t](#) pad, [uint32_t](#) val)
This function writes a configuration register.

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.

14.6.1 Detailed Description

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

14.7 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_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.
- #define `SC_PM_RESET_REASON_V2X_DEBUG` 13U
V2X debug switch.

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_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_partition_wake](#) ([sc_ipc_t](#) ipc, [sc_rm_pt_t](#) pt)
This function sends a wake interrupt to 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 specifies 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_get_reset_part` (`sc_ipc_t` ipc, `sc_rm_pt_t` *pt)
This function gets the partition that caused a reset.
- `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.
- `sc_err_t sc_pm_set_boot_parm` (`sc_ipc_t` ipc, `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 change the boot parameters for 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_reboot_continue` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt)
This function is used to continue the 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.
- `void sc_pm_cpu_reset` (`sc_ipc_t` ipc, `sc_rsrc_t` resource, `sc_faddr_t` address)
This function is used to reset a CPU.
- `sc_err_t sc_pm_resource_reset` (`sc_ipc_t` ipc, `sc_rsrc_t` resource)
This function is used to reset a peripheral.
- `sc_bool_t sc_pm_is_partition_started` (`sc_ipc_t` ipc, `sc_rm_pt_t` pt)
This function returns a bool indicating if a partition was started.

14.7.1 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.

14.8 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`.
- #define [SC_RM_DET_W](#) 1U
Width of `sc_rm_det_t`.
- #define [SC_RM_RMSG_W](#) 4U
Width of `sc_rm_rmsg_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.

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.
- typedef [uint8_t sc_rm_det_t](#)
This type is used to indicate memory region transactions should detour to the IEE.
- typedef [uint8_t sc_rm_rmsg_t](#)
This type is used to assign an RMSG value to a memory region.

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.

- `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_err_t sc_rm_get_resource_owner` (`sc_ipc_t ipc`, `sc_rsrc_t resource`, `sc_rm_pt_t *pt`)

This function is used to get the owner 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 an existing memory region.

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

This function requests that the SC fragment 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_err_t sc_rm_set_memreg_iee` (`sc_ipc_t ipc`, `sc_rm_mr_t mr`, `sc_rm_det_t det`, `sc_rm_rmsg_t rmsg`)

This function configures the IEE parameters 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.

14.8.1 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.

14.9 platform/svc/seco/api.h File Reference

Header file containing the public API for the System Controller (SC) Security (SECO) function.

Macros

Defines for `sc_seco_auth_cmd_t`

- `#define SC_SECO_AUTH_CONTAINER 0U`
Authenticate container.
- `#define SC_SECO_VERIFY_IMAGE 1U`
Verify image.
- `#define SC_SECO_REL_CONTAINER 2U`
Release container.
- `#define SC_SECO_AUTH_SECO_FW 3U`
SECO Firmware.
- `#define SC_SECO_AUTH_HDMI_TX_FW 4U`
HDMI TX Firmware.
- `#define SC_SECO_AUTH_HDMI_RX_FW 5U`
HDMI RX Firmware.
- `#define SC_SECO_EVERIFY_IMAGE 6U`
Enhanced verify image.

Defines for `seco_rng_stat_t`

- `#define SC_SECO_RNG_STAT_UNAVAILABLE 0U`
Unable to initialize the RNG.
- `#define SC_SECO_RNG_STAT_INPROGRESS 1U`
Initialization is on-going.
- `#define SC_SECO_RNG_STAT_READY 2U`
Initialized.

Typedefs

- typedef [uint8_t sc_seco_auth_cmd_t](#)
This type is used to issue SECO authenticate commands.
- typedef [uint32_t sc_seco_rng_stat_t](#)
This type is used to return the RNG initialization status.

Functions

Image Functions

- [sc_err_t sc_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_seco_authenticate](#) ([sc_ipc_t](#) ipc, [sc_seco_auth_cmd_t](#) cmd, [sc_faddr_t](#) addr)
This function is used to authenticate a SECO image or command.
- [sc_err_t sc_seco_enh_authenticate](#) ([sc_ipc_t](#) ipc, [sc_seco_auth_cmd_t](#) cmd, [sc_faddr_t](#) addr, [uint32_t](#) mask1, [uint32_t](#) mask2)
This function is used to authenticate a SECO image or command.

Lifecycle Functions

- [sc_err_t sc_seco_forward_lifecycle](#) ([sc_ipc_t](#) ipc, [uint32_t](#) change)
This function updates the lifecycle of the device.
- [sc_err_t sc_seco_return_lifecycle](#) ([sc_ipc_t](#) ipc, [sc_faddr_t](#) addr)
This function updates the lifecycle to one of the return lifecycles.
- [sc_err_t sc_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.

Attestation Functions

- [sc_err_t sc_seco_attest_mode](#) ([sc_ipc_t](#) ipc, [uint32_t](#) mode)
This function is used to set the attestation mode.
- [sc_err_t sc_seco_attest](#) ([sc_ipc_t](#) ipc, [uint64_t](#) nonce)
This function is used to request attestation.
- [sc_err_t sc_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_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_seco_attest_verify](#) ([sc_ipc_t](#) ipc, [sc_faddr_t](#) addr)
This function is used to verify attestation.

Key Functions

- [sc_err_t sc_seco_gen_key_blob](#) ([sc_ipc_t](#) ipc, [uint32_t](#) id, [sc_faddr_t](#) load_addr, [sc_faddr_t](#) export_addr, [uint16_t](#) max_size)
This function is used to generate a SECO key blob.
- [sc_err_t sc_seco_load_key](#) ([sc_ipc_t](#) ipc, [uint32_t](#) id, [sc_faddr_t](#) addr)
This function is used to load a SECO key.

Manufacturing Protection Functions

- `sc_err_t sc_seco_get_mp_key` (sc_ipc_t ipc, sc_faddr_t dst_addr, uint16_t dst_size)
This function is used to get the manufacturing protection public key.
- `sc_err_t sc_seco_update_mpmr` (sc_ipc_t ipc, sc_faddr_t addr, uint8_t size, uint8_t lock)
This function is used to update the manufacturing protection message register.
- `sc_err_t sc_seco_get_mp_sign` (sc_ipc_t ipc, sc_faddr_t msg_addr, uint16_t msg_size, sc_faddr_t dst_addr, uint16_t dst_size)
This function is used to get the manufacturing protection signature.

Debug Functions

- void `sc_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_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_seco_enable_debug` (sc_ipc_t ipc, sc_faddr_t addr)
This function securely enables debug.
- `sc_err_t sc_seco_get_event` (sc_ipc_t ipc, uint8_t idx, uint32_t *event)
This function is used to return an event from the SECO error log.

Miscellaneous Functions

- `sc_err_t sc_seco_fuse_write` (sc_ipc_t ipc, sc_faddr_t addr)
This function securely writes a group of fuse words.
- `sc_err_t sc_seco_patch` (sc_ipc_t ipc, sc_faddr_t addr)
This function applies a patch.
- `sc_err_t sc_seco_set_mono_counter_partition` (sc_ipc_t ipc, uint16_t *she)
This function partitions the monotonic counter.
- `sc_err_t sc_seco_set_fips_mode` (sc_ipc_t ipc, uint8_t mode, uint32_t *reason)
This function configures the SECO in FIPS mode.
- `sc_err_t sc_seco_start_rng` (sc_ipc_t ipc, sc_seco_rng_stat_t *status)
This function starts the random number generator.
- `sc_err_t sc_seco_sab_msg` (sc_ipc_t ipc, sc_faddr_t addr)
This function sends a generic signed message to the SECO SHE/HSM components.
- `sc_err_t sc_seco_secvio_enable` (sc_ipc_t ipc)
This function is used to enable security violation and tamper interrupts.
- `sc_err_t sc_seco_secvio_config` (sc_ipc_t ipc, uint8_t id, uint8_t access, uint32_t *data0, uint32_t *data1, uint32_t *data2, uint32_t *data3, uint32_t *data4, uint8_t size)
This function is used to read/write SNVS security violation and tamper registers.
- `sc_err_t sc_seco_secvio_dgo_config` (sc_ipc_t ipc, uint8_t id, uint8_t access, uint32_t *data)
This function is used to read/write SNVS security violation and tamper DGO registers.

14.9.1 Detailed Description

Header file containing the public API for the System Controller (SC) Security (SECO) function.

14.10 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

Watchdog 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_set_wdog_window` (`sc_ipc_t` ipc, `sc_timer_wdog_time_t` window)
This function sets the watchdog window 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.

14.10.1 Detailed Description

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

Index

ipc.h

- sc_ipc_close, [234](#)
- sc_ipc_open, [234](#)
- sc_ipc_read, [234](#)
- sc_ipc_write, [235](#)

IRQ: Interrupt Service, [99](#)

- sc_irq_enable, [102](#)
- sc_irq_status, [102](#)

MISC: Miscellaneous Service, [104](#)

- sc_misc_api_ver, [110](#)
- sc_misc_board_ioctl, [117](#)
- sc_misc_boot_done, [112](#)
- sc_misc_boot_status, [111](#)
- sc_misc_build_info, [109](#)
- sc_misc_debug_out, [109](#)
- sc_misc_get_boot_container, [116](#)
- sc_misc_get_boot_dev, [115](#)
- sc_misc_get_boot_type, [115](#)
- sc_misc_get_button_status, [116](#)
- sc_misc_get_control, [107](#)
- sc_misc_get_temp, [114](#)
- sc_misc_otp_fuse_read, [112](#)
- sc_misc_otp_fuse_write, [113](#)
- sc_misc_rompatch_checksum, [117](#)
- sc_misc_set_ari, [111](#)
- sc_misc_set_control, [106](#)
- sc_misc_set_dma_group, [108](#)
- sc_misc_set_max_dma_group, [107](#)
- sc_misc_set_temp, [113](#)
- sc_misc_unique_id, [110](#)
- sc_misc_waveform_capture, [109](#)

PAD: Pad Service, [118](#)

- sc_pad_28fdsoi_dse_t, [123](#)
- sc_pad_28fdsoi_ps_t, [123](#)
- sc_pad_28fdsoi_pus_t, [123](#)
- sc_pad_config, [130](#)
- sc_pad_config_t, [123](#)
- sc_pad_get, [129](#)
- sc_pad_get_all, [128](#)
- sc_pad_get_gp, [125](#)
- sc_pad_get_gp_28fdsoi, [131](#)
- sc_pad_get_gp_28fdsoi_comp, [134](#)
- sc_pad_get_gp_28fdsoi_hsic, [132](#)
- sc_pad_get_mux, [124](#)

- sc_pad_get_wakeup, [126](#)

- sc_pad_iso_t, [123](#)

- sc_pad_set, [129](#)

- sc_pad_set_all, [127](#)

- sc_pad_set_gp, [125](#)

- sc_pad_set_gp_28fdsoi, [130](#)

- sc_pad_set_gp_28fdsoi_comp, [133](#)

- sc_pad_set_gp_28fdsoi_hsic, [132](#)

- sc_pad_set_mux, [123](#)

- sc_pad_set_wakeup, [126](#)

- platform/config/mx8qx/pads.h, [227](#)

- platform/main/ipc.h, [233](#)

- platform/main/types.h, [235](#)

- platform/svc/irq/api.h, [253](#)

- platform/svc/misc/api.h, [255](#)

- platform/svc/pad/api.h, [258](#)

- platform/svc/pm/api.h, [261](#)

- platform/svc/rm/api.h, [266](#)

- platform/svc/seco/api.h, [269](#)

- platform/svc/timer/api.h, [271](#)

PM: Power Management Service, [136](#)

- sc_pm_boot, [154](#)

- sc_pm_clock_enable, [150](#)

- sc_pm_cpu_reset, [157](#)

- sc_pm_cpu_start, [157](#)

- sc_pm_get_clock_parent, [151](#)

- sc_pm_get_clock_rate, [149](#)

- sc_pm_get_reset_part, [153](#)

- sc_pm_get_resource_power_mode, [145](#)

- sc_pm_get_sys_power_mode, [143](#)

- sc_pm_is_partition_started, [159](#)

- sc_pm_partition_wake, [143](#)

- sc_pm_power_mode_t, [141](#)

- sc_pm_reboot, [155](#)

- sc_pm_reboot_continue, [156](#)

- sc_pm_reboot_partition, [156](#)

- sc_pm_req_cpu_low_power_mode, [146](#)

- sc_pm_req_low_power_mode, [146](#)

- sc_pm_req_sys_if_power_mode, [148](#)

- sc_pm_reset, [152](#)

- sc_pm_reset_reason, [153](#)

- sc_pm_resource_reset, [158](#)

- sc_pm_set_boot_parm, [154](#)

- sc_pm_set_clock_parent, [151](#)

- sc_pm_set_clock_rate, [149](#)

- sc_pm_set_cpu_resume, [148](#)
- sc_pm_set_cpu_resume_addr, [147](#)
- sc_pm_set_partition_power_mode, [142](#)
- sc_pm_set_resource_power_mode, [144](#)
- sc_pm_set_resource_power_mode_all, [145](#)
- sc_pm_set_sys_power_mode, [142](#)
- RM: Resource Management Service, [160](#)
 - sc_rm_assign_memreg, [182](#)
 - sc_rm_assign_pad, [185](#)
 - sc_rm_assign_resource, [172](#)
 - sc_rm_dump, [187](#)
 - sc_rm_find_memreg, [182](#)
 - sc_rm_get.did, [168](#)
 - sc_rm_get_memreg_info, [185](#)
 - sc_rm_get_partition, [170](#)
 - sc_rm_get_resource_info, [178](#)
 - sc_rm_get_resource_owner, [177](#)
 - sc_rm_is_memreg_owned, [184](#)
 - sc_rm_is_pad_owned, [187](#)
 - sc_rm_is_resource_master, [177](#)
 - sc_rm_is_resource_owned, [176](#)
 - sc_rm_is_resource_peripheral, [178](#)
 - sc_rm_memreg_alloc, [179](#)
 - sc_rm_memreg_frag, [180](#)
 - sc_rm_memreg_free, [181](#)
 - sc_rm_memreg_split, [180](#)
 - sc_rm_move_all, [171](#)
 - sc_rm_partition_alloc, [166](#)
 - sc_rm_partition_free, [168](#)
 - sc_rm_partition_lock, [169](#)
 - sc_rm_partition_static, [169](#)
 - sc_rm_perm_t, [166](#)
 - sc_rm_rmsg_t, [166](#)
 - sc_rm_set_confidential, [167](#)
 - sc_rm_set_master_attributes, [174](#)
 - sc_rm_set_master_sid, [175](#)
 - sc_rm_set_memreg_iee, [184](#)
 - sc_rm_set_memreg_permissions, [183](#)
 - sc_rm_set_pad_movable, [186](#)
 - sc_rm_set_parent, [170](#)
 - sc_rm_set_peripheral_permissions, [175](#)
 - sc_rm_set_resource_movable, [173](#)
 - sc_rm_set_subsys_rsrc_movable, [173](#)
- sc_ipc_close
 - ipc.h, [234](#)
- sc_ipc_open
 - ipc.h, [234](#)
- sc_ipc_read
 - ipc.h, [234](#)
- sc_ipc_write
 - ipc.h, [235](#)
- sc_irq_enable
 - IRQ: Interrupt Service, [102](#)
- sc_irq_status
 - IRQ: Interrupt Service, [102](#)
- sc_misc_api_ver
 - MISC: Miscellaneous Service, [110](#)
- sc_misc_board_ioctl
 - MISC: Miscellaneous Service, [117](#)
- sc_misc_boot_done
 - MISC: Miscellaneous Service, [112](#)
- sc_misc_boot_status
 - MISC: Miscellaneous Service, [111](#)
- sc_misc_build_info
 - MISC: Miscellaneous Service, [109](#)
- sc_misc_debug_out
 - MISC: Miscellaneous Service, [109](#)
- sc_misc_get_boot_container
 - MISC: Miscellaneous Service, [116](#)
- sc_misc_get_boot_dev
 - MISC: Miscellaneous Service, [115](#)
- sc_misc_get_boot_type
 - MISC: Miscellaneous Service, [115](#)
- sc_misc_get_button_status
 - MISC: Miscellaneous Service, [116](#)
- sc_misc_get_control
 - MISC: Miscellaneous Service, [107](#)
- sc_misc_get_temp
 - MISC: Miscellaneous Service, [114](#)
- sc_misc_otf_fuse_read
 - MISC: Miscellaneous Service, [112](#)
- sc_misc_otf_fuse_write
 - MISC: Miscellaneous Service, [113](#)
- sc_misc_rompatch_checksum
 - MISC: Miscellaneous Service, [117](#)
- sc_misc_set_ari
 - MISC: Miscellaneous Service, [111](#)
- sc_misc_set_control
 - MISC: Miscellaneous Service, [106](#)
- sc_misc_set_dma_group
 - MISC: Miscellaneous Service, [108](#)
- sc_misc_set_max_dma_group
 - MISC: Miscellaneous Service, [107](#)
- sc_misc_set_temp
 - MISC: Miscellaneous Service, [113](#)
- sc_misc_unique_id
 - MISC: Miscellaneous Service, [110](#)
- sc_misc_waveform_capture
 - MISC: Miscellaneous Service, [109](#)
- SC_P_ALL
 - types.h, [252](#)
- sc_pad_28fdsoi_dse_t
 - PAD: Pad Service, [123](#)
- sc_pad_28fdsoi_ps_t
 - PAD: Pad Service, [123](#)
- sc_pad_28fdsoi_pus_t
 - PAD: Pad Service, [123](#)

- sc_pad_config
 - PAD: Pad Service, [130](#)
- sc_pad_config_t
 - PAD: Pad Service, [123](#)
- sc_pad_get
 - PAD: Pad Service, [129](#)
- sc_pad_get_all
 - PAD: Pad Service, [128](#)
- sc_pad_get_gp
 - PAD: Pad Service, [125](#)
- sc_pad_get_gp_28fdsoi
 - PAD: Pad Service, [131](#)
- sc_pad_get_gp_28fdsoi_comp
 - PAD: Pad Service, [134](#)
- sc_pad_get_gp_28fdsoi_hsic
 - PAD: Pad Service, [132](#)
- sc_pad_get_mux
 - PAD: Pad Service, [124](#)
- sc_pad_get_wakeup
 - PAD: Pad Service, [126](#)
- sc_pad_iso_t
 - PAD: Pad Service, [123](#)
- sc_pad_set
 - PAD: Pad Service, [129](#)
- sc_pad_set_all
 - PAD: Pad Service, [127](#)
- sc_pad_set_gp
 - PAD: Pad Service, [125](#)
- sc_pad_set_gp_28fdsoi
 - PAD: Pad Service, [130](#)
- sc_pad_set_gp_28fdsoi_comp
 - PAD: Pad Service, [133](#)
- sc_pad_set_gp_28fdsoi_hsic
 - PAD: Pad Service, [132](#)
- sc_pad_set_mux
 - PAD: Pad Service, [123](#)
- sc_pad_set_wakeup
 - PAD: Pad Service, [126](#)
- sc_pad_t
 - types.h, [253](#)
- sc_pm_boot
 - PM: Power Management Service, [154](#)
- sc_pm_clock_enable
 - PM: Power Management Service, [150](#)
- sc_pm_cpu_reset
 - PM: Power Management Service, [157](#)
- sc_pm_cpu_start
 - PM: Power Management Service, [157](#)
- sc_pm_get_clock_parent
 - PM: Power Management Service, [151](#)
- sc_pm_get_clock_rate
 - PM: Power Management Service, [149](#)
- sc_pm_get_reset_part
 - PM: Power Management Service, [153](#)
- sc_pm_get_resource_power_mode
 - PM: Power Management Service, [145](#)
- sc_pm_get_sys_power_mode
 - PM: Power Management Service, [143](#)
- sc_pm_is_partition_started
 - PM: Power Management Service, [159](#)
- sc_pm_partition_wake
 - PM: Power Management Service, [143](#)
- sc_pm_power_mode_t
 - PM: Power Management Service, [141](#)
- sc_pm_reboot
 - PM: Power Management Service, [155](#)
- sc_pm_reboot_continue
 - PM: Power Management Service, [156](#)
- sc_pm_reboot_partition
 - PM: Power Management Service, [156](#)
- sc_pm_req_cpu_low_power_mode
 - PM: Power Management Service, [146](#)
- sc_pm_req_low_power_mode
 - PM: Power Management Service, [146](#)
- sc_pm_req_sys_if_power_mode
 - PM: Power Management Service, [148](#)
- sc_pm_reset
 - PM: Power Management Service, [152](#)
- sc_pm_reset_reason
 - PM: Power Management Service, [153](#)
- sc_pm_resource_reset
 - PM: Power Management Service, [158](#)
- sc_pm_set_boot_parm
 - PM: Power Management Service, [154](#)
- sc_pm_set_clock_parent
 - PM: Power Management Service, [151](#)
- sc_pm_set_clock_rate
 - PM: Power Management Service, [149](#)
- sc_pm_set_cpu_resume
 - PM: Power Management Service, [148](#)
- sc_pm_set_cpu_resume_addr
 - PM: Power Management Service, [147](#)
- sc_pm_set_partition_power_mode
 - PM: Power Management Service, [142](#)
- sc_pm_set_resource_power_mode
 - PM: Power Management Service, [144](#)
- sc_pm_set_resource_power_mode_all
 - PM: Power Management Service, [145](#)
- sc_pm_set_sys_power_mode
 - PM: Power Management Service, [142](#)
- SC_R_NONE
 - types.h, [252](#)
- sc_rm_assign_memreg
 - RM: Resource Management Service, [182](#)
- sc_rm_assign_pad
 - RM: Resource Management Service, [185](#)
- sc_rm_assign_resource
 - RM: Resource Management Service, [172](#)

- sc_rm_dump
 - RM: Resource Management Service, [187](#)
- sc_rm_find_memreg
 - RM: Resource Management Service, [182](#)
- sc_rm_get_did
 - RM: Resource Management Service, [168](#)
- sc_rm_get_memreg_info
 - RM: Resource Management Service, [185](#)
- sc_rm_get_partition
 - RM: Resource Management Service, [170](#)
- sc_rm_get_resource_info
 - RM: Resource Management Service, [178](#)
- sc_rm_get_resource_owner
 - RM: Resource Management Service, [177](#)
- sc_rm_is_memreg_owned
 - RM: Resource Management Service, [184](#)
- sc_rm_is_pad_owned
 - RM: Resource Management Service, [187](#)
- sc_rm_is_resource_master
 - RM: Resource Management Service, [177](#)
- sc_rm_is_resource_owned
 - RM: Resource Management Service, [176](#)
- sc_rm_is_resource_peripheral
 - RM: Resource Management Service, [178](#)
- sc_rm_memreg_alloc
 - RM: Resource Management Service, [179](#)
- sc_rm_memreg_frag
 - RM: Resource Management Service, [180](#)
- sc_rm_memreg_free
 - RM: Resource Management Service, [181](#)
- sc_rm_memreg_split
 - RM: Resource Management Service, [180](#)
- sc_rm_move_all
 - RM: Resource Management Service, [171](#)
- sc_rm_partition_alloc
 - RM: Resource Management Service, [166](#)
- sc_rm_partition_free
 - RM: Resource Management Service, [168](#)
- sc_rm_partition_lock
 - RM: Resource Management Service, [169](#)
- sc_rm_partition_static
 - RM: Resource Management Service, [169](#)
- sc_rm_perm_t
 - RM: Resource Management Service, [166](#)
- sc_rm_rmsg_t
 - RM: Resource Management Service, [166](#)
- sc_rm_set_confidential
 - RM: Resource Management Service, [167](#)
- sc_rm_set_master_attributes
 - RM: Resource Management Service, [174](#)
- sc_rm_set_master_sid
 - RM: Resource Management Service, [175](#)
- sc_rm_set_memreg_iee
 - RM: Resource Management Service, [184](#)
- sc_rm_set_memreg_permissions
 - RM: Resource Management Service, [183](#)
- sc_rm_set_pad_movable
 - RM: Resource Management Service, [186](#)
- sc_rm_set_parent
 - RM: Resource Management Service, [170](#)
- sc_rm_set_peripheral_permissions
 - RM: Resource Management Service, [175](#)
- sc_rm_set_resource_movable
 - RM: Resource Management Service, [173](#)
- sc_rm_set_subsys_rsrc_movable
 - RM: Resource Management Service, [173](#)
- sc_rsrc_t
 - types.h, [252](#)
- sc_seco_attest
 - SECO: Security Service, [198](#)
- sc_seco_attest_mode
 - SECO: Security Service, [197](#)
- sc_seco_attest_verify
 - SECO: Security Service, [200](#)
- sc_seco_authenticate
 - SECO: Security Service, [193](#)
- sc_seco_build_info
 - SECO: Security Service, [204](#)
- sc_seco_chip_info
 - SECO: Security Service, [205](#)
- sc_seco_commit
 - SECO: Security Service, [196](#)
- sc_seco_enable_debug
 - SECO: Security Service, [205](#)
- sc_seco_enh_authenticate
 - SECO: Security Service, [194](#)
- sc_seco_forward_lifecycle
 - SECO: Security Service, [195](#)
- sc_seco_fuse_write
 - SECO: Security Service, [207](#)
- sc_seco_gen_key_blob
 - SECO: Security Service, [201](#)
- sc_seco_get_attest_pkey
 - SECO: Security Service, [198](#)
- sc_seco_get_attest_sign
 - SECO: Security Service, [199](#)
- sc_seco_get_event
 - SECO: Security Service, [206](#)
- sc_seco_get_mp_key
 - SECO: Security Service, [202](#)
- sc_seco_get_mp_sign
 - SECO: Security Service, [204](#)
- sc_seco_image_load
 - SECO: Security Service, [192](#)
- sc_seco_load_key
 - SECO: Security Service, [201](#)
- sc_seco_patch
 - SECO: Security Service, [207](#)

- sc_seco_return_lifecycle
 - SECO: Security Service, [196](#)
- sc_seco_sab_msg
 - SECO: Security Service, [210](#)
- sc_seco_secvio_config
 - SECO: Security Service, [212](#)
- sc_seco_secvio_dgo_config
 - SECO: Security Service, [212](#)
- sc_seco_secvio_enable
 - SECO: Security Service, [211](#)
- sc_seco_set_fips_mode
 - SECO: Security Service, [209](#)
- sc_seco_set_mono_counter_partition
 - SECO: Security Service, [208](#)
- sc_seco_start_rng
 - SECO: Security Service, [210](#)
- sc_seco_update_mpmr
 - SECO: Security Service, [203](#)
- sc_timer_cancel_rtc_alarm
 - TIMER: Timer Service, [223](#)
- sc_timer_cancel_sysctr_alarm
 - TIMER: Timer Service, [225](#)
- sc_timer_get_rtc_sec1970
 - TIMER: Timer Service, [221](#)
- sc_timer_get_rtc_time
 - TIMER: Timer Service, [221](#)
- sc_timer_get_wdog_status
 - TIMER: Timer Service, [218](#)
- sc_timer_ping_wdog
 - TIMER: Timer Service, [218](#)
- sc_timer_pt_get_wdog_status
 - TIMER: Timer Service, [219](#)
- sc_timer_set_rtc_alarm
 - TIMER: Timer Service, [222](#)
- sc_timer_set_rtc_calb
 - TIMER: Timer Service, [223](#)
- sc_timer_set_rtc_periodic_alarm
 - TIMER: Timer Service, [222](#)
- sc_timer_set_rtc_time
 - TIMER: Timer Service, [220](#)
- sc_timer_set_sysctr_alarm
 - TIMER: Timer Service, [224](#)
- sc_timer_set_sysctr_periodic_alarm
 - TIMER: Timer Service, [224](#)
- sc_timer_set_wdog_action
 - TIMER: Timer Service, [219](#)
- sc_timer_set_wdog_pre_timeout
 - TIMER: Timer Service, [216](#)
- sc_timer_set_wdog_timeout
 - TIMER: Timer Service, [215](#)
- sc_timer_set_wdog_window
 - TIMER: Timer Service, [216](#)
- sc_timer_start_wdog
 - TIMER: Timer Service, [217](#)
- sc_timer_stop_wdog
 - TIMER: Timer Service, [217](#)
- SECO: Security Service, [188](#)
 - sc_seco_attest, [198](#)
 - sc_seco_attest_mode, [197](#)
 - sc_seco_attest_verify, [200](#)
 - sc_seco_authenticate, [193](#)
 - sc_seco_build_info, [204](#)
 - sc_seco_chip_info, [205](#)
 - sc_seco_commit, [196](#)
 - sc_seco_enable_debug, [205](#)
 - sc_seco_enh_authenticate, [194](#)
 - sc_seco_forward_lifecycle, [195](#)
 - sc_seco_fuse_write, [207](#)
 - sc_seco_gen_key_blob, [201](#)
 - sc_seco_get_attest_pkey, [198](#)
 - sc_seco_get_attest_sign, [199](#)
 - sc_seco_get_event, [206](#)
 - sc_seco_get_mp_key, [202](#)
 - sc_seco_get_mp_sign, [204](#)
 - sc_seco_image_load, [192](#)
 - sc_seco_load_key, [201](#)
 - sc_seco_patch, [207](#)
 - sc_seco_return_lifecycle, [196](#)
 - sc_seco_sab_msg, [210](#)
 - sc_seco_secvio_config, [212](#)
 - sc_seco_secvio_dgo_config, [212](#)
 - sc_seco_secvio_enable, [211](#)
 - sc_seco_set_fips_mode, [209](#)
 - sc_seco_set_mono_counter_partition, [208](#)
 - sc_seco_start_rng, [210](#)
 - sc_seco_update_mpmr, [203](#)
- TIMER: Timer Service, [214](#)
 - sc_timer_cancel_rtc_alarm, [223](#)
 - sc_timer_cancel_sysctr_alarm, [225](#)
 - sc_timer_get_rtc_sec1970, [221](#)
 - sc_timer_get_rtc_time, [221](#)
 - sc_timer_get_wdog_status, [218](#)
 - sc_timer_ping_wdog, [218](#)
 - sc_timer_pt_get_wdog_status, [219](#)
 - sc_timer_set_rtc_alarm, [222](#)
 - sc_timer_set_rtc_calb, [223](#)
 - sc_timer_set_rtc_periodic_alarm, [222](#)
 - sc_timer_set_rtc_time, [220](#)
 - sc_timer_set_sysctr_alarm, [224](#)
 - sc_timer_set_sysctr_periodic_alarm, [224](#)
 - sc_timer_set_wdog_action, [219](#)
 - sc_timer_set_wdog_pre_timeout, [216](#)
 - sc_timer_set_wdog_timeout, [215](#)
 - sc_timer_set_wdog_window, [216](#)
 - sc_timer_start_wdog, [217](#)
 - sc_timer_stop_wdog, [217](#)
- types.h

SC_P_ALL, [252](#)
sc_pad_t, [253](#)
SC_R_NONE, [252](#)
sc_rsrc_t, [252](#)